FreeBSD® JOURNAL

January/February 2020

Network Research and Standardization

Jail vnet by Examples

Getting Started with FreeBSD/RISC-V

Groundbreaking TrueNAS® M Series

DISRUPTING STORAGE AT WARP SPEED

LOWEST TCO SHARED STORAGE

Intel® Xeon® Scalable Family Processors

RESILIENT

- Self-healing Data
- Continuous Operation
- Easy Replication

WARP FACTOR IX

- Faster than SSD-based Hybrid Storage Arrays
- Flash-Turbocharged Data Access

EXPANDABLE

- Scales to 10PB using HGST Drives
- 10Gb/s-100Gb/s per NAS
- Non-disruptive Upgrades

COMPATIBLE

- Citrix, Veeam, & VMware Certified
- Unifies File, Block, & S3 Data
- Supports Leading Cloud Providers

Visit **iXsystems.com/TrueNAS** or call **(855) GREP-4-iX** today!

HGST a Western Digital brand

iXsystems™

© 2018 iXsystems. TrueNAS is a registered trademark of iXsystems, Inc. All rights reserved. Intel, the Intel logo, Xeon, and Xeon Inside are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries. All product names, logos, and brands are property of their respective owners.

FreeBSD® JOURNAL
Editorial Board

John Baldwin • FreeBSD Developer, Member of the FreeBSD Core Team and Chair of *FreeBSD Journal* Editorial Board.

Bryan Drewery • Senior Software Engineer at EMC Isilon, member of FreeBSD Portmgr Team, and FreeBSD Committer.

Justin Gibbs • Founder of the FreeBSD Foundation, President of the FreeBSD Foundation, and a Software Engineer at Facebook.

Daichi Goto • Director at BSD Consulting Inc. (Tokyo).

Joseph Kong • Software Development Engineer at Amazon Web Services, author of *FreeBSD Device Drivers* and *Designing BSD Rootkits.*

Dru Lavigne • Director of Storage Engineering at iXsystems, author of *BSD Hacks* and *The Best of FreeBSD Basics.*

Michael W Lucas • Author of *Absolute FreeBSD.*

Ed Maste • Director of Project Development, FreeBSD Foundation.

Kirk McKusick • Treasurer of the FreeBSD Foundation Board, and lead author of *The Design and Implementation* book series.

George V. Neville-Neil • Director of the FreeBSD Foundation Board, and co-author of *The Design and Implementation of the FreeBSD Operating System.*

Philip Paeps • Secretary of the FreeBSD Foundation Board, FreeBSD Committer, and Independent Consultant.

Kristof Provost • Treasurer of the EuroBSDCon Foundation, FreeBSD Committer, and Independent Consultant.

Hiroki Sato • Director of the FreeBSD Foundation Board, Chair of Asia BSDCon, member of the FreeBSD Core Team, and Assistant Professor at Tokyo Institute of Technology.

Benedict Reuschling • Vice President of the FreeBSD Foundation Board, a FreeBSD Documentation Committer, and member of the FreeBSD Core Team.

Robert N. M. Watson • Director of the FreeBSD Foundation Board, Founder of the TrustedBSD Project, and University Senior Lecturer at the University of Cambridge.

S&W PUBLISHING LLC
PO BOX 408, BELFAST, MAINE 04915

Publisher • Walter Andrzejewski
walter@freebsdjournal.com

Editor-at-Large • James Maurer
jmaurer@freebsdjournal.com

Copy Editor • Annaliese Jakimides

Design & Production • Reuter & Associates

Advertising Sales • Walter Andrzejewski
walter@freebsdjournal.com
Call 888/290-9469

FreeBSD Journal (ISBN: 978-0-615-88479-0) is published 6 times a year (January/February, March/April, May/June, July/August, September/October, November/December).
Published by the FreeBSD Foundation,
3980 Broadway St. STE #103-107, Boulder, CO 80304
ph: 720/207-5142 • fax: 720/222-2350
email: info@freebsdfoundation.org
Copyright © 2020 by FreeBSD Foundation. All rights reserved. This magazine may not be reproduced in whole or in part without written permission from the publisher.

LETTER from the Board

Celebrating One Year of Free Access

Dear Readers,

The January/February 2020 edition marks the first anniversary of the move to making the *FreeBSD Journal* a free publication. As you may remember, the Foundation made the decision to change the subscription model in order to help increase awareness and adoption of FreeBSD. Twelve months later, the benefit of making it freely accessible to everyone still outweighs the cost of producing it. The *FreeBSD Journal* is an excellent venue for keeping those interested in FreeBSD up-to-date.

In addition to making the *Journal* freely available, we also made the decision to remove it from the App stores. It allows us to have more control of the publication. While we removed the apps from the various storefronts last year, it has come to our attention that app subscribers may still be receiving subscription renewal notices via the app store. Unfortunately, these are out of our control. We ask that you please ignore any subscription renewal notices you may receive.

As always, thank you for your continued support of the *FreeBSD Journal*. Please consider making a donation to help us support this effort. Every little bit helps. We look forward to bringing you *FreeBSD Journal*'s timely and informative articles in each issue and hope you'll share your favorites with colleagues and friends.

Anne Dickison
FreeBSD Foundation

FreeBSD® JOURNAL
January/February 2020

12 **Getting Started with FreeBSD/RISC-V**
The lack of availability on the hardware side and incomplete support on the software side mean that RISC-V's usefulness is, at present, mainly limited to research and specialized computing work. *By Mitchell Horne*

19 **Jail vnet by Examples**
The Virtual Network feature (vnet) allows each jail to have its own routing table/ ARD & NDP cache and interfaces. *By Olivier Cochard-Labbé*

35 **Network Research and Standardization**
FreeBSD has a long lineage in the research community as both a target and research vehicle. This research looks at Internet protocols and how they can be improved to make the Internet better. *By Tom Jones*

41 **Interview**
In this October 2019 interview with **Trenton Schulz**, *Allan Jude* and *Benedict Reuschling* talk with him about his early days with FreeBSD, the Robot Operating System (ROS), Qt, and more.

3 **Foundation Letter**
Celebrating One Year of Free Access *By Anne Dickison*

5 **New Faces of FreeBSD**
In this installment, the spotlight is on **Loïc Bartoletti**, who received his ports bit in January. *By Dru Lavigne*

7 **FreeBSD Foundation Report**
From advocating for FreeBSD to implementing FreeBSD features, Deb Goodkin's team has been there to help make FreeBSD the best open-source project and operating system out there. *By Deb Goodkin*

9 **We Get Letters**
What the heck does "Research" mean in an operating system anyway?
By Michael W Lucas

48 **Events Calendar**
By Dru Lavigne and *Anne Dickison*

new faces of FreeBSD

BY DRU LAVIGNE

This column shines a spotlight on contributors who recently received their commit bit and introduces them to the FreeBSD community. In this installment, the spotlight is on Loïc Bartoletti, who received his ports bit in January.

Tell us a bit about yourself, your background, and your interests.

• Loïc: With a background in history and urban planning, I have experienced a professional evolution from being a user of analysis tools to a creator of the tools, particularly the graphics/qgis, databases/grass, and databases/postgis ports. I am now a GIS/CAD engineer and a C/C++/Python/SQL developer who is learning Nim / Rust and Qt for UI. My professional subjects are GIS CAD Land survey GNSS (GPS) databases using PostgreSQL.

I am personally and professionally involved in the OpenSource Geo movement (OSGEO), and I make sure that the tools that are developed can be used on FreeBSD. This involves correcting codes to integrate upstream BSD, creating ports for our system, and promoting them.

The list of ports that I maintain reflects my professional and personal activity.

How did you first learn about FreeBSD and what about FreeBSD interested you?

• Loïc: My first experience was 2004/2005 to test a "UNIX" system. I first heard about Linux, but it didn't work, or it worked very badly, on my hardware. Coming from Apple, I had, of course, also heard about FreeBSD. By chance, I found a magazine in French with a CD and very good documentation on how to install it. I really liked it at the time: its ease of use, stability, separation between base and packages.

I continued watching BSD out of intellectual curiosity until I became a regular user of FreeBSD 6 (around 2007/2008) at home. Since 2018, by progressively integrating FreeBSD in a VM on my Windows workstation, I have used FreeBSD daily in my new job.

How did you end up becoming a committer?

• Loïc: Over a few months, I became a little more involved in the Project, both in my contributions and in my relationships with other contributors, especially #kde-freebsd. And finally, Tobias suggested my name to join the ports team.

How has your experience been since joining the FreeBSD Project? Do you have any advice for readers who may be interested in also becoming FreeBSD committers?

• Loïc: It's really-early days, so for now I don't have enough hindsight, but I was amazed at the number of messages I received when I introduced myself to the other developers.

I continue my learning, I try to polish my ports, and I look to help the other maintainers as well, while being well supported by my two mentors.

I don't have any particular advice, but I think it's essential to communicate with others and to be part of the teams that interest you.

DRU LAVIGNE is a FreeBSD doc committer and the author of *BSD Hacks* and *The Best of FreeBSD Basics*.

Jails ARE FBSD'S MOST LEGENDARY FEATURE:

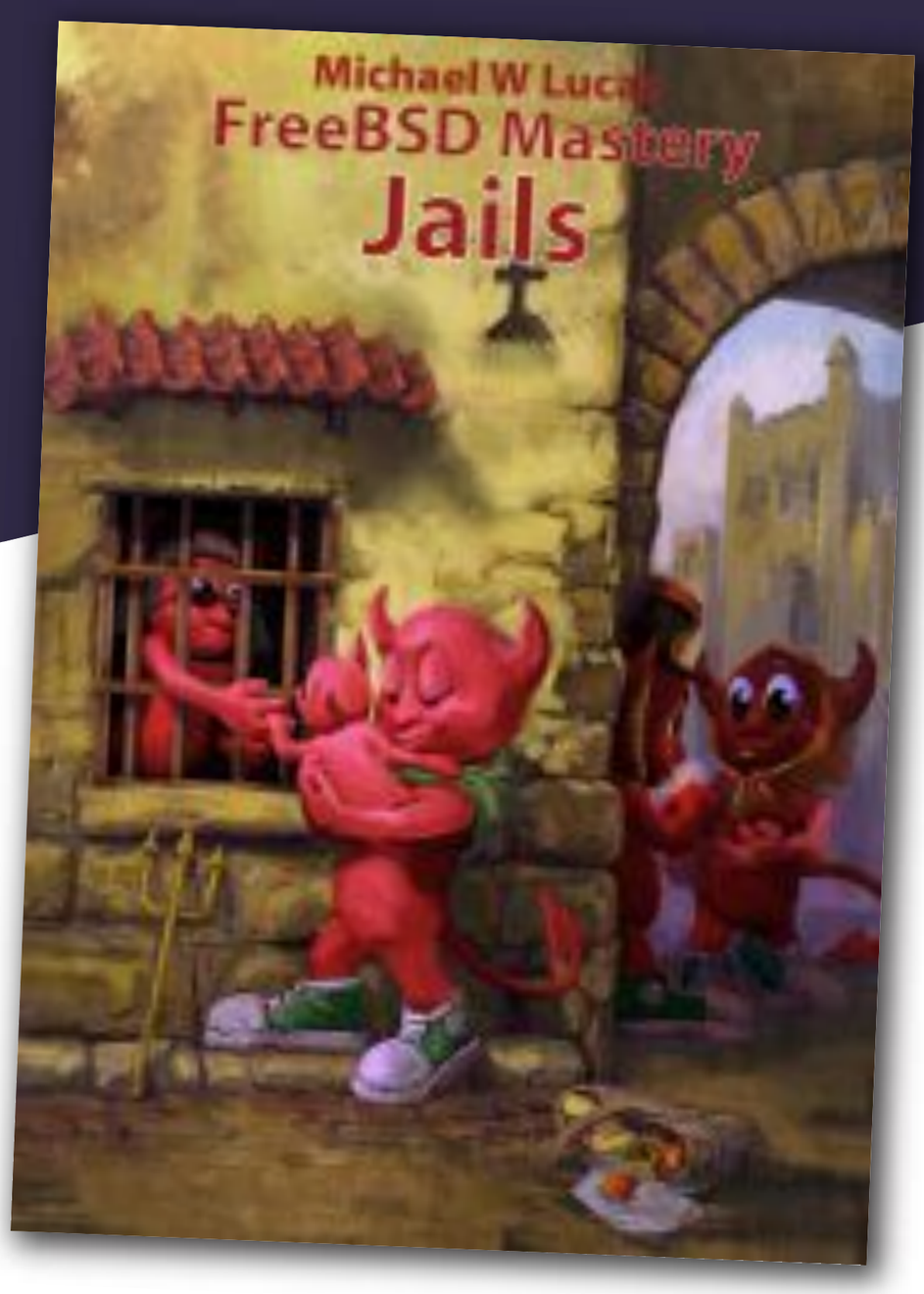
KNOWN TO BE POWERFUL, TRICKY TO MASTER, AND CLOAKED IN DECADES OF DUBIOUS LORE.

FreeBSD Mastery: Jails cuts through the clutter to expose the inner mechanisms of jails and unleash their power in your service.

Confine Your Software!

- Understand how jails achieve lightweight virtualization
- Understand the base system's jail tools and the iocage toolkit
- Optimally configure hardware
- Manage jails from the host and from within the jail
- Optimize disk space usage to support thousands of jails
- Comfortably work within the limits of jails
- Implement fine-grained control of jail features
- Build virtual networks
- Deploy hierarchical jails
- Constrain jail resource usage.

…And much, much more!



FreeBSD Mastery: Jails BY MICHAEL W LUCAS ***Available at All Bookstores***

Foundation REPORT

BY DEB GOODKIN

Now that 2020 is upon us, I wanted to take a minute to share with *Journal* readers what an amazing year 2019 was for supporting the FreeBSD Project and community! Why do I say that? Because as I reflect over the last year, I realize how many events we've attended all over the world and how many lives we've touched in so many ways. From advocating for FreeBSD to implementing FreeBSD features, my team has been there to help make FreeBSD the best open-source project and operating system out there.

In 2019, we focused on supporting a few key areas where the Project needed the most help. The first area was software development. Whether it was contracting FreeBSD developers to work on projects like WiFi support or providing internal staff to quickly implement hardware work-arounds, we've stepped in to help keep FreeBSD innovative, secure, and reliable. Software development includes supporting the tools and infrastructure that make the development process go smoothly, and we're on it with team members heading up the continuous integration efforts, and we're actively involved in the clusteradmin and security teams. In fact, last year we committed over 1,200 code changes; reviewed and provided feedback on hundreds of other developer changes, and improved the continuous integration test pass rate by almost 100%! Read more about our Software Development Projects and CI and Testing Advancements.

Our advocacy efforts focused on recruiting new users and contributors to the Project. We attended and participated in 38 conferences and events in 21 countries. From giving FreeBSD presentations and workshops to staffing tables, we were able to have 1:1 conversations with thousands of attendees. You can see a full recap of our advocacy efforts here.

Our travels also provided opportunities to talk directly with FreeBSD commercial and individual users, contributors, and future FreeBSD user/contributors. We've seen an increase in use and interest in FreeBSD from all of these organizations and individuals. These meetings give us a chance to learn more about what organizations need and what they and other individuals are working on. The information helps inform the work we should fund.

"We've stepped in to help keep FreeBSD innovative, secure, and reliable."

Last year also marked a significant change for the *FreeBSD Journal*. The Foundation chose to make it a free publication. As you can see from above, a big part of the FreeBSD Foundation's mission is to raise awareness of FreeBSD throughout the world. The *FreeBSD Journal* is an excellent venue for keeping those interested in FreeBSD up-to-date. The articles are timely and informative. Therefore, the Foundation decided that the benefit of making it freely accessible to everyone outweighs the cost of producing it.

In 2019, your donations helped us continue our efforts to support critical areas of FreeBSD such as:

- Operating System Improvements: providing staff to immediately respond to urgent problems and implement new features and functionality allowing for the innovation and stability you've come to rely on.
- Improving and increasing test coverage, continuous integration, and automated testing with a full-time software engineer to ensure you receive the highest-quality, secure, and reliable operating system.
- Security: providing engineering resources to bolster the capacity and responsiveness of the security team, giving you peace of mind when security issues arise.

- Growing the number of FreeBSD contributors and users from our global FreeBSD outreach and advocacy efforts, including expansion into regions such as China, India, Africa, and Singapore.
- Offering FreeBSD workshops and presentations at more conferences, meetups, and universities around the world.
- Providing opportunities such as developer and vendor summits and company visits to help facilitate collaboration between commercial users and FreeBSD developers, as well as helping to get changes pushed into the FreeBSD source tree and creating a bigger and healthier ecosystem.
- Providing the professionally published *FreeBSD Journal* for anyone and everyone wanting to stay informed about the latest FreeBSD developments.

We've accomplished a lot this year, but we are still only a small 501(c)(3) organization focused on supporting FreeBSD and not a trade organization like many other open-source foundations.

For us to continue and increase our efforts for FreeBSD we depend 100% on your donations. We need your help to continue and increase the work we are doing. Your support directly impacts the Project.

If you love FreeBSD like we do, please help us spread the word and, if you haven't already done so, please make a donation today. Together we can accomplish even more for FreeBSD.

DEB GOODKIN is the Executive Director of the FreeBSD Foundation. She's thrilled to be in her 15th year at the Foundation and is proud of her hardworking and dedicated team. She spent over 20 years in the data storage industry in engineering development, technical sales, and technical marketing. When not working, you'll find her on her road or mountain bike, running, hiking with her dogs, skiing the slopes of Colorado, or reading a good book.

Thank you!

The FreesBSD Foundation would like to acknowledge the following companies for their continued support of the Project. Because of generous donations such as these we are able to continue moving the Project forward.

FreeBSD™ FOUNDATION™

Are you a fan of FreeBSD? Help us give back to the Project and donate today! **freebsdfoundation.org/donate/**

Please check out the full list of generous community investors at freebsdfoundation.org/donors/

Uranium

intel

Iridium

NetApp

Platinum

NETFLIX

Gold

facebook JUNIPER NETWORKS

Silver

BECKHOFF Microsoft

STORMSHIELD

Tarsnap vmware

WeGet letters

by Michael W Lucas

Dear Letter-Answering Entity,

What the heck does "Research" mean anyway in an operating system?

—Perplexed

Good day, Perplexed,

How are you? How's the family? What's it like, living a life where you can send random questions nobody else in the history of computing would bother themselves with off into the void with a complete disregard for the social niceties and expect someone you've never even met to spend their precious time answering?

Me and my stupid honor. Why did I agree to do this column again?

But to answer your question, let's start with the Single Source of Truth for the English language, my 1933 *Oxford English Dictionary*, all our linguistic wisdom distilled into 13 weighty tomes that even carry the aroma of enlightenment, except, of course, for all words beginning with "Rz" because of the dire nature of those inimical vocables. Truly, we owe those exalted few who compiled the slender (and tightly secured) 14th volume a debt of honor we can never repay because they've all passed on. I'm not implying that Rz* dispassionately eradicated them in much the same way we might exterminate termites, of course. I would never say any such thing where they might possibly hear.

So, let's look at the definition of research.

dial. form of Rice1.

No, wait. That's "Ryze," the last word in the Poy-Ry volume. How anyone can pick up a dictionary and view only one word completely baffles me. Such people are not to be trusted. Let's try again. There are several definitions, but the one that seems most common is:

To search again and repeatedly.

We all do that! I've even gone searching for the solution to a technical problem and discovered a mailing list post from the decadent age of the last century where I declare that I've searched all over for a solution.

Nobody answered, of course. If they knew the answer, it would be in the archives. Younger Me never answered that message to explain what was going on. Jerk.

But a more interesting definition is:

A search or investigation directed to the discovery of some fact by careful consideration or study of a subject; a course of critical or scientific inquiry.

Many computer people think that they're scientists when truly they're science fans or, worse, disguise their biases and antipathies by loudly declaring them to be science. Did you do legitimate statistical analysis of your data from the last decade, including graphs and means and the population's standard deviance of sample correlation? Did you even retain that data in the first place? If not, you're no different from the dude watching American football who sprawls on his couch yelling at the television that he would have done a tackle on the last play to sink the eight ball past the other team's wicket. Stop pretending that your weak-kneed

science fandom is on a level with people who earned doctorates and got grants and perform actual math-and-measuring science. That's as annoying as the kid who loves computers thinking his enthusiasm is as powerful as your hard-earned knowledge and sweat-drenched experience.

You can keep the lab coat. Nobody minds when fans cosplay.

Legit science isn't a result, or a paper, or using math. You can't disbelieve science or declare it's not relevant, because science is a process. Not believing in science is like not believing in walking. It exists. Science has four parts: observation, hypothesis, experiment, and results. In the 400 years this method has been used, we've gone from riding rivers to riding rockets, from burning wood to burning the whole planet. Real science is undeniably potent. It deserves your fandom.

So, you look at the world. You observe a bunch of details.

You make a guess as to why something happens the way it does. When you can state that guess clearly and succinctly, you get to call it a hypothesis.

You can't prove the hypothesis is correct, but you can prove it's wrong. You figure out a practical way to do so and perform the test.

If the test shows the hypothesis is wrong, great! You know a little more than you used to. If the test shows you might be right, that's nice too. Remember, it's not you that's wrong. It's the hypothesis. And hypotheses are intended to be spawned and discarded like processes. Getting emotionally involved with a hypothesis is like being attached to your web server running at PID 691. Even if you hard-code that process ID into the kernel, it'll distort everything around it and unnecessarily complicate your life.

Either way, write up your results and tell people about them. Yes, even when you're wrong.

Billions of iterations of this process gave us cat videos, effective cancer treatments, and nostick cookware.

Computers can fit anywhere into this process. Maybe you're observing your computer and throwing a tantrum when it misbehaves. Perhaps you're using the computer to do some math to see if your first guess is even plausible. Computers make possible tests that our predecessors couldn't even imagine. And if nothing else, you'll probably use a computer to analyze and publish your results.

You need an operating system that works predictably and reliably. Something that you fully control, rather than relying on dubiously documented updates imposed by an OS manufacturer. You need an OS that you can customize to support your labors.

If you're reading this column, you know what I recommend.

But what I'll also recommend to you?

Science.

Don't just run computers or write code. Observe the results. Measure the impact of changes. When I started with Unix, we had DBX and shell scripts running ping(1) and were delighted beyond all reason to have them. Today we have more monitoring tools than you can charge a rhino at. Software like DTrace makes poking at system internals easier than ever. We use them, but only in a limited occasional way.

Track what your systems do.

See what wobbles.

Observe behavior changes when you apply patches, or install that new switch, or tweak that bit of kernel code.

Make a hypothesis.

Test the hypothesis.

Document the tests.

Working with the scientific method demands not only math, but (gasp) statistics. Statistics determine if your observations or results are meaningful.

Then document your results. Even if the results disprove your hypothesis. And you can apply this to the simplest parts of the computing profession.

- Observations: The server keeps rebooting unexpectedly. Armadillos are nesting in the server case.
- Hypothesis: The armadillos are rebooting the server.
- Test: If I remove the armadillos and the unexpected reboots continue, my hypothesis is proven false.
- Results: I removed the armadillos. The reboots continued. The hypothesis is false. Also, I developed leprosy.

This is research. This is science. I highly commend it to you.

Have a question for Michael?
Send it to letters@freebsdjournal.org

MICHAEL W LUCAS (https://mwl.io)'s newest books are *Sudo Mastery, 2nd Edition* and *Terrapin Sky Tango.*

Write For Us!

Contact Jim Maurer with your article ideas.

(**jmaurer@freebsdjournal.com**)

Getting Started with FreeBSD/RISC-V

BY MITCHELL HORNE

Introducing FreeBSD's newest CPU architecture: RISC-V. This article will bring you up to speed on what you need to know about the architecture and how you can build and run your very own FreeBSD/RISC-V system using QEMU.

Introduction to the RISC-V Architecture

RISC-V is an open and extendable ISA (Instruction-Set Architecture) developed by UC Berkeley over the last decade. It is a RISC architecture that was designed to learn from some of the pitfalls and mistakes made by existing computer architectures.

RISC-V has two main claims to fame. The first is that the ISA's specifications are released as open source under a BSD license, which means that RISC-V is completely free to use in both commercial and academic contexts. This, combined with the fact that it can be implemented quite simply, makes RISC-V attractive for use in academia, both for research implementations and as a tool for teaching computer architecture. For Silicon companies looking to avoid the heavy licensing fees they might pay for other architectures such as ARM, RISC-V provides a tempting cost-free alternative. Finally, there are projects like lowRISC (https://www.lowrisc.org/our-work/) and OpenHW Group's CORE-V (https://www.openhwgroup.org/news/2019/12/10/openhw-group-announces-core-v-chassis-soc-project-and-issues-industry-call-for-participation/) that aim to produce fully open-source, Unix-capable, RISC-V-based SoCs. This will make it possible to run systems that are completely free and open-source—from the OS and applications running on it, all the way down to the CPU cores themselves.

The RISC-V instruction set is also modular by design. The idea here is that a RISC-V implementation requires only a small number of integer instructions (the base ISA), and depending on the desired use-case, more instructions can be added in the form of official or unofficial ex-

tensions. As a surprising example, multiplication and division instructions are not required by the base integer instruction set. Instead, they come as part of the official "M" extension. The thinking is that a small, single-core, embedded core might not require this functionality, so by making it optional, the implementer is given the freedom to omit it. More complicated SMP systems might require additional functionality like atomics or hardware floating-point, and these are provided as official extensions as well. The "G" (general) extension includes several extensions necessary for running most Unix-like operating systems. The hope of the RISC-V designers is that by making the RISC-V freely available and easily extended, it can be adopted for a wide range of use cases—from the smallest microprocessors to large multi-CPU server platforms.

FreeBSD and RISC-V

RISC-V is FreeBSD's most recent and experimental supported architecture. Work on this port began in 2015, led by Ruslan Bukin (br@). In 2016, it was officially imported into the FreeBSD source tree. Currently, FreeBSD's RISC-V support is classified as Tier-3, which means that it is still considered to be under development. As a result, no guarantees are made about feature availability, ABI stability, or support from the security, ports, or release engineering teams. This is not to say that the RISC-V port is unusable; on the contrary, the last few years have seen slow but steady improvements, and the majority of the base system is fully functional.

In particular, the RISC-V hardware ecosystem is still young, and Unix-capable, RISC-V SoCs are not readily available. FreeBSD supports SiFive's HiFive Unleashed, one of the few boards of this type on the market today, but its high price and lack of availability make it an impractical choice for most consumers. For now, much of the development and testing of FreeBSD/RISC-V is done using simulators such as QEMU or Spike. As RISC-V matures and its adoption increases, there will be opportunities for FreeBSD's support for it to improve as well.

Building a FreeBSD/RISC-V Image

All right, that's enough information for now, so let's get to the fun part. We will build a 64-bit RISC-V system from source. This can be done from an amd64 host running any supported version of FreeBSD.

First, we must obtain a RISC-V toolchain. This includes the cross-compiler, linker, and other utilities required to build the FreeBSD operating system from source. We will be using the GNU toolchain since it has the most mature RISC-V support at the moment. You can install the RISC-V GNU toolchain using `pkg(8)`:

```
pkg install riscv64-xtoolchain-gcc
```

This will install the `devel/riscv64-binutils` and `devel/riscv64-gcc` packages. This preconfigured toolchain should be everything you need to cross-build FreeBSD. All of FreeBSD/RISC-V's sources are in HEAD, so, using your favorite version control utility, grab a copy of FreeBSD's sources and start building:

```
git checkout https://github.com/freebsd/freebsd.git freebsd-riscv
cd freebsd-riscv

# First, build the userland libraries and utilities
make -j4 CROSS_TOOLCHAIN=riscv64-gcc TARGET=riscv buildworld
```

```
# Next, the kernel. We're building the QEMU kernel config.
make -j4 CROSS_TOOLCHAIN=riscv64-gcc TARGET=riscv KERNCONF=QEMU buildkernel
```

Optionally, you can edit `sys/riscv/conf/QEMU` before compiling and set `ROOTDEVNAME=/dev/vtbd0p1` line to point to the correct root filesystem for this setup.

Note: for those who are a little adventurous, FreeBSD's in-tree version of clang and lld should have the necessary support to build FreeBSD/RISC-V, if you're running CURRENT. Try it out with `make -j4 TARGET=riscv buildworld`, *but your mileage may vary!*

Wait a little time (or a long time) until compilation has finished. If all went well, then you can move on to the next step. We want to install the newly built FreeBSD/RISC-V root filesystem to some user-accessible directory so we can later generate an image file. You can specify the directory you want with the `DESTDIR` make variable.

```
# NO_ROOT allows us to install files as a regular user.
make TARGET=riscv -DNO_ROOT DESTDIR=$HOME/riscv-root installworld
make TARGET=riscv -DNO_ROOT DESTDIR=$HOME/riscv-root distribution
make TARGET=riscv -DNO_ROOT DESTDIR=$HOME/riscv-root installkernel
```

Now that we've installed all the necessary files, we want to create a disk image that can be read by QEMU. We will first generate a ufs root filesystem using `makefs(8)`, and then create the image including a swap partition using `mkimg(1)`.

Change to the root filesystem directory we just populated. You can use the following script to generate the image.

```
#!/bin/sh

# Create /etc/rc.conf and append to METALOG
echo 'hostname="qemu"' > etc/rc.conf
s=$(($(cat etc/rc.conf | wc -c)))
echo "./etc/rc.conf type=file uname=root gname=wheel mode=0644 size=$s" >> METALOG

# Create /etc/fstab and append to METALOG
echo "/dev/vtbd0p1 / ufs rw,noatime 1 1" > etc/fstab
echo "/dev/vtbd0p2 / swapsw  0 0" >> etc/fstab
s=$(($(cat etc/fstab | wc -c)))
echo "./etc/fstab type=file uname=root gname=wheel mode=0644 size=$s" >> METALOG

# Create FreeBSD ufs root partition
makefs -D -B little \
    -o label=freebsd_root \
    -o version=2 \
    -s 20g -f 65% \
    riscvroot.ufs METALOG

# Create the final .img
mkimg -s gpt -p freebsd-ufs:=riscvroot.ufs -p freebsd-swap::4G -o riscvroot.img
```

Booting FreeBSD

Now that we have built our own FreeBSD image, it's time to test it out. As mentioned, we will be using QEMU. Please install the `emulators/qemu-devel` package from ports. Note that the regular `emulators/qemu` is enough to run FreeBSD, but there have been many improvements to the RISC-V QEMU platforms, so `emulators/qemu-devel` is recommended.

You will also need to install OpenSBI via `sysutils/opensbi`. OpenSBI will act as the bootloader, and it provides low-level firmware functions required by the FreeBSD kernel via its Supervisor Binary Interface (SBI).

With OpenSBI installed, you can now boot FreeBSD using the following command:

```
qemu-system-riscv64 -machine virt -smp 2 -m 2G -nographic \
     -kernel $HOME/riscv-root/boot/kernel/kernel \
     -bios /usr/local/share/opensbi/platform/qemu/virt/firmware/fw_jump.elf \
     -drive file=/path/to/riscvroot.img,format=raw,id=hd0 \
     -device virtio-blk-device,drive=hd0
```

If everything went well, you should see FreeBSD begin to start up. If you skipped the optional step earlier, the system will fail to mount the root filesystem. When that happens, simply enter `ufs:/dev/vtbd0p1` at the prompt.

You might be surprised not to see the familiar `loader(8)` prompt. This is because `loader(8)` has not yet been ported to RISC-V, and so the kernel boots directly from OpenSBI. For now, you won't have the benefit of loader tweaks or tunables, but this will certainly be available in the future.

You should see the system boot to the login prompt. Log in as root, and change the root password with `passwd(1)`.

Networking

Most systems aren't all that useful without a network connection, so let's power-off and fix that. Add the following arguments to the commandline when launching QEMU:

```
-netdev user,hostfwd=tcp::10000-:22,id=net0 -device virtio-net-device,netdev=net0
```

As you can see, we are forwarding TCP port 10000 from the host machine to port 22 of the guest. Port 22 is the default port used by `ssh(1)`. Before we can connect, we must enable `sshd(8)` on the guest by appending the following to `/etc/rc.conf`:

```
sshd_enable="YES"
```

Now, start the `sshd(8)` service with:

```
service sshd start
```

You should now be able to log in to your QEMU guest; just provide the proper port.

```
ssh -p 10000 mhorne@localhost
```

If you have a `tap(4)` device bridged to a real network card, then you can use that too. Append the following arguments to QEMU instead:

```
-netdev tap,ifname=tap0,script=no,id=net1 -device virtio-net-device,netdev=net1
```

A `tap(4)` interface will allow your guest to appear to the rest of your network as any other host would. Connect using the IP address as it appears in the output of `ifconfig(8)` on the guest:

```
ssh mhorne@10.0.1.25
```

Updating Your System

Let's say some time has gone by, and you want to update your FreeBSD/RISC-V system to take advantage of some new fix or feature that just landed in HEAD. For a typical FreeBSD machine, you might do a self-hosted build, i.e., build and install using the same machine you are updating. Unfortunately, since QEMU is being used to run FreeBSD/RISC-V, we're at the mercy of the emulator, and that means that any self-hosted build will be SLOW.

You could, of course, follow the steps from the previous sections after updating the source tree, and you would end up with a fresh, new FreeBSD/RISC-V system; but, any configuration you've done would be lost. Let's look at how you might update the system within an existing root image file.

First, we will perform the same steps as before to cross-compile an updated version of FreeBSD, targeted for riscv64.

```
# Update the source tree, assuming git
cd freebsd-riscv
git pull

# Now, rebuild and install
make -j4 CROSS_TOOLCHAIN=riscv64-gcc TARGET=riscv buildworld
make -j4 CROSS_TOOLCHAIN=riscv64-gcc TARGET=riscv KERNCONF=QEMU buildkernel
```

Now the updated system is ready to be installed. However, instead of installing to some destination directory on the host, this time we want to update the contents of the image file that we generated earlier. We can make use of `mdconfig(8)`, which will allow us to create and mount our image file as a memory disk.

Note: This, and the steps that follow, require superuser privilege. To prevent concurrent access to the filesystem, please ensure you have shut down any instance of QEMU that is using the image file before continuing.

```
# Create the memory device
mdconfig -a -f /path/to/riscvroot.img
```

The name of the new memory device will be output to the console. Remember that the image we generated contains two partitions: a ufs root partition and a swap partition. We want to mount the ufs partition so that we can install the updated system there.

```
# For memory device /dev/md0
mount /dev/md0p1 /mnt

# Install the system to the mounted partition
make TARGET=riscv DESTDIR=/mnt installworld
make TARGET=riscv DESTDIR=/mnt installkernel

# Optionally delete stale files
make TARGET=riscv DESTDIR=/mnt delete-old
make TARGET=riscv DESTDIR=/mnt delete-old-libs

# Now, unmount
umount /dev/md0p1
```

And you're done! You can boot up the freshly updated system using the same image file you created before. One thing to note is that the kernel specified on the QEMU commandline is the one that will be booted, rather than the one installed to the image filesystem. Therefore, you might need to search for it in the `/usr/obj` directory or do an `installkernel` to a local folder on the host.

Conclusion

The intent of this article is to introduce the RISC-V CPU architecture and to allow easy experimentation with FreeBSD/RISC-V. Hopefully you now feel confident in your ability to build, install, and run your own FreeBSD/RISC-V system. Interested users are encouraged to continue tracking changes and updating their system, now that they have it set up. For further reading, check out the RISC-V page on the FreeBSD wiki (https://wiki.freebsd.org/riscv), and subscribe to the freebsd-riscv mailing list (https://lists.freebsd.org/mailman/listinfo/freebsd-riscv).

Certainly, it is still early days for RISC-V as a whole. The lack of availability on the hardware side and incomplete support on the software side mean that RISC-V's usefulness is, at present, mainly limited to research and specialized computing work. The many improvements to software support and increased adoption of RISC-V over the last few years signify that this will not always be the case and that it may soon emerge as a more accessible platform for general computing. As RISC-V continues to grow,

---

MITCHELL HORNE is a university student currently finishing his undergrad at the University of Waterloo, in Canada. He is a FreeBSD src committer and over the last year has become one of the main contributors to the FreeBSD/RISC-V port.

FreeBSD® JOURNAL

The FreeBSD Journal is Now Free!

Yep, that's right Free.

The voice of the FreeBSD Community and the BEST way to keep up with the latest releases and new developments in FreeBSD is now openly available to everyone.

DON'T MISS A SINGLE ISSUE!

2019 Editorial Calendar:

- Getting Started with FreeBSD (Jan/Feb 2019)
- Debugging and Testing (March/April 2019)
- FreeBSD for Makers (May/June 2019)
- Containerization (July/Aug 2019)
- Security (Sept/Oct 2019)
- Network Virtualization (Nov/Dec 2019)

Find out more at: freebsdfoundation/journal

Jail vnet by Examples

BY OLIVIER COCHARD-LABBÉ

To understand the Virtual Network feature (vnet), not to be confused with vtnet(4) for VirtIO Ethernet driver, let's begin with an extract from the vnet(9) man page:

```
DESCRIPTION
     vnet is the name of a technique to virtualize the network stack.
     (...).
     Each (virtual) network stack is attached to a prison, with vnet0
     being the unrestricted default network stack of the base system.
```

As a related prison feature, let's check the jail(8) man page section about vnet:

```
vnet     Create the jail with its own virtual network stack, with its own
         network interfaces, addresses, routing table, etc.  The kernel
         must have been compiled with the VIMAGE option for this to be
         available.  Possible values are "inherit" to use the system
         network stack, possibly with restricted IP addresses, and "new"
         to create a new network stack.
```

To resume, it's a feature that allows each jail to have its own routing table / ARD & NDP cache and interfaces

Vocabulary

- Host: The system hosting your jails

Examples

These examples use "empty" jails based on the host '/' to focus only on the vnet feature. They will all be in "persist" mode (because no processes are running).

Concerning the OS requirements:

- The shell used is /bin/sh.
- FreeBSD 12.1 minimum (can be 12-STABLE or even better a -head)

Useless Isolated vnet Jail

This useless example shows how to create an isolated vnet jail.

Command line parameters details:

- -c: create a new jail
- name: Name of the jail to avoid using its jail ID (JID) later
- host.hostname: For this example, only used to have a pretty output with command 'jls'
- persist: There is no process running on this jail, so force it running
- vnet: Enable the virtual network stack

```
# jail -c name=useless host.hostname=jvnet persist vnet
# jls
   JID  IP Address      Hostname                      Path
     1                  jvnet                         /
```

Here is the default network interface assigned and the content of its routing table for this new jail.

```
# jexec useless ifconfig
lo0: flags=8008<LOOPBACK,MULTICAST> metric 0 mtu 16384
        options=680003<RXCSUM,TXCSUM,LINKSTATE,RXCSUM_IPV6,TXCSUM_IPV6>
        groups: lo
        nd6 options=21<PERFORMNUD,AUTO_LINKLOCAL>
# jexec useless netstat -rn
Routing tables
```

There is an unconfigured loopback (disabled and no IP addresses assigned) and an empty routing table. Let's fix that.

```
# jexec useless service netif restart
Stopping Network: lo0.
lo0: flags=8008<LOOPBACK,MULTICAST> metric 0 mtu 16384
        options=680003<RXCSUM,TXCSUM,LINKSTATE,RXCSUM_IPV6,TXCSUM_IPV6>
        groups: lo
        nd6 options=21<PERFORMNUD,AUTO_LINKLOCAL>
Starting Network: lo0.
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> metric 0 mtu 16384
        options=680003<RXCSUM,TXCSUM,LINKSTATE,RXCSUM_IPV6,TXCSUM_IPV6>
        inet6 ::1 prefixlen 128
        inet6 fe80::1%lo0 prefixlen 64 scopeid 0x1
        inet 127.0.0.1 netmask 0xff000000
        groups: lo
        nd6 options=21<PERFORMNUD,AUTO_LINKLOCAL>
# jexec useless netstat -rn
Routing tables

Internet:
Destination        Gateway            Flags      Netif Expire
127.0.0.1          link#1             UH           lo0
```

```
Internet6:
Destination                       Gateway                       Flags     Netif Expire
::1                               link#1                        UH          lo0
fe80::%lo0/64                     link#1                        U           lo0
fe80::1%lo0                       link#1                        UHS         lo0
```

This is a lot better! But we only have the loopback interface running. The next step is to create a virtual Ethernet tap interface and assign it to the jail. The ifconfig(8) man page extract:

```
vnet jail
        Move the interface to the jail(8), specified by name or JID.  If
        the jail has a virtual network stack, the interface will
        disappear from the current environment and become visible to the
        jail.
```

Let's do this:

```
# TAP=$(ifconfig tap create)
# ifconfig $TAP
tap0: flags=8802<BROADCAST,SIMPLEX,MULTICAST> metric 0 mtu 1500
        options=80000<LINKSTATE>
        ether 00:bd:70:98:00:00
        groups: tap
        media: Ethernet autoselect
        status: no carrier
        nd6 options=29<PERFORMNUD,IFDISABLED,AUTO_LINKLOCAL>
# ifconfig $TAP vnet useless
# ifconfig $TAP
ifconfig: interface tap0 does not exist
```

What happened? Just after we enabled the interface and assigned it to the jail, it disappeared! That is the expected behavior because this interface doesn't belong to your host network stack anymore. You can check its status on the jail and even assign an IP address to it:

```
# jexec useless ifconfig $TAP
tap0: flags=8802<BROADCAST,SIMPLEX,MULTICAST> metric 0 mtu 1500
        options=80000<LINKSTATE>
        ether 00:bd:70:98:00:00
        groups: tap
        media: Ethernet autoselect
        status: no carrier
        nd6 options=21<PERFORMNUD,AUTO_LINKLOCAL>
# jexec useless ifconfig $TAP inet 192.0.2.1/24 up
# jexec useless ifconfig $TAP inet
tap0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
        options=80000<LINKSTATE>
        inet 192.0.2.1 netmask 0xffffff00 broadcast 192.0.2.255
# jexec useless ping -c 2 192.0.2.1
```

```
PING 192.0.2.1 (192.0.2.1): 56 data bytes
64 bytes from 192.0.2.1: icmp_seq=0 ttl=64 time=0.248 ms
64 bytes from 192.0.2.1: icmp_seq=1 ttl=64 time=0.525 ms

--- 192.0.2.1 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.248/0.387/0.525/0.139 ms
# jexec useless arp -na
? (192.0.2.1) at 00:bd:70:98:00:00 on tap0 permanent [ethernet]
# arp -na | grep 192.0.2.1
#
```

The jail can ping its own interface, its own ARP cache is populated with the corresponding entry, and all of these are isolated from the host networking stack.

It's the same for the routing table:

```
# jexec useless route add -net 198.51.100.0/24 192.0.2.2
add net 198.51.100.0: gateway 192.0.2.2
# jexec useless netstat -4rn
Routing tables

Internet:
Destination        Gateway            Flags     Netif Expire
127.0.0.1          link#1             UH          lo0
192.0.2.0/24       link#2             U          tap0
192.0.2.1          link#2             UHS         lo0
198.51.100.0/24    192.0.2.2          UGS        tap0
# netstat -4rn | grep 198.51.100.0
#
```

Before continuing to the next example, we will clean up the existing jail and destroy the tap interface. We need to use the -R (upper case) option to remove a jail created without a configuration file. Using option -r (lower case), the vnet interface will not be removed to the host automatically.

```
# jail -R useless
# ifconfig $TAP destroy
```

vnet Jail Connected to the Host

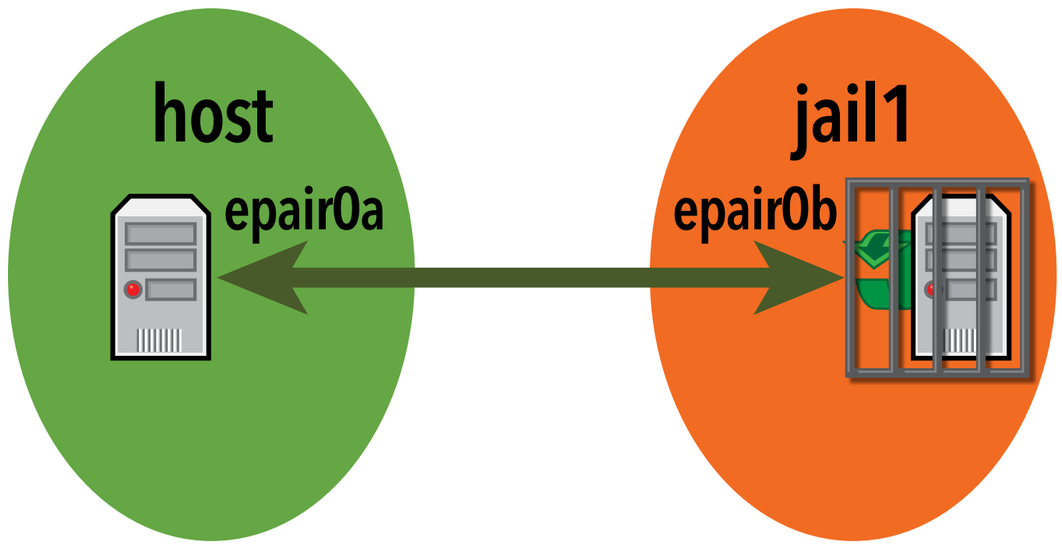
This example shows how to communicate between the jail and the host itself. The main problem is that any interface put into a vnet will disappear from the host networking stack.

So, we could imagine this setup:

1. Create a bridge and assign an IP address to it
2. Create a tap interface and add it into the bridge
3. Assign the tap interface to the jvnet jail

But this will not work. The TAP interface will move from the host (=disappear from the host) to join the jail's vnet, and so it will disappear from the bridge too!

To solve this problem, the epair(4) interface (Ethernet pair) was created. This special network interface represents two interfaces (epairXa and epairXb) that will behave like two Ethernet interfaces cross-connected between them. By assigning each side to a different vnet, they will still exchange frames between them.



Start by creating a new epair pair.

```
# ifconfig epair create
epair0a
# ifconfig -g epair
epair0b
epair0a
```

The host is showing its two new interfaces: epair0a and epair0b.
Create a new jail, named “jvnet” and assign interface epair0b to it.

```
# jail -c name=jvnet host.hostname=jvnet persist vnet vnet.interface=epair0b
```

Interface epair0b no longer belongs to the host system network stack, but the other epair0a still does! Let’s configure an IP address on epair0a.

```
# ifconfig -g epair
epair0a
# ifconfig epair0a inet 192.0.2.1/24 up
```

Then do the same on epair0b belonging to the jail and check their connectivity.

```
# jexec jvnet ifconfig epair0b inet 192.0.2.2/24 up
# ping -c 2 192.0.2.2
PING 192.0.2.2 (192.0.2.2): 56 data bytes
64 bytes from 192.0.2.2: icmp_seq=0 ttl=64 time=0.285 ms
64 bytes from 192.0.2.2: icmp_seq=1 ttl=64 time=0.532 ms

--- 192.0.2.2 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.285/0.408/0.532/0.124 ms
# arp -na | grep 192.0.2.
? (192.0.2.2) at 02:77:29:04:9e:0b on epair0a expires in 1139 seconds [ethernet]
? (192.0.2.1) at 02:77:29:04:9e:0a on epair0a permanent [ethernet]
```

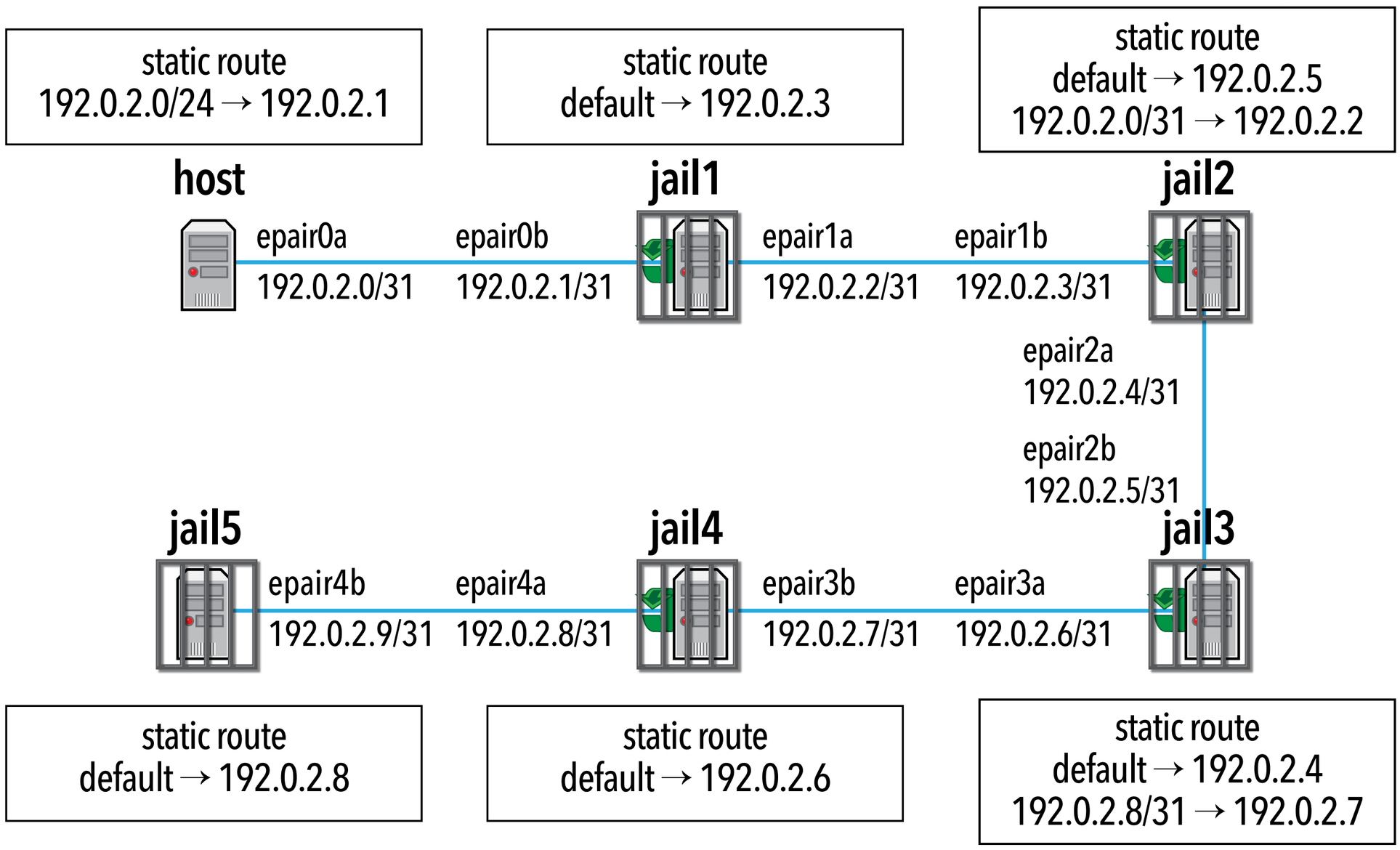
By displaying their MAC address, you will notice that epair are using specific MAC address. Destroy this jail and the epair pair interfaces before continuing to the next example:

```
# jail -R jvnet
# ifconfig epair0a destroy
```

Now that we've got a basic setup working, let's complexify a little more using multiple jails configured in serial and with routing between them.

Chained Routed vnet Jails

In this example we will create four jails to route traffic from the host toward a fifth jail.



Start to generate the five epairs with a first loop.

```
# for i in $(jot 5); do ifconfig epair create; done
epair0a
epair1a
epair2a
epair3a
epair4a
```

Then generate five jails with epair assigned to them with a mix of loop and manual assignment.

```
# for i in $(jot 4); do jail -c name=hop$i host.hostname=hop$i persist vnet \
vnet.interface=epair$((i-1))b vnet.interface=epair${i}a; done
# jail -c name=hop5 host.hostname=hop5 persist vnet vnet.interface=epair4b
# jls
   JID  IP Address      Hostname                      Path
     3                  hop1                          /
     4                  hop2                          /
```

```
5               hop3                    /
6               hop4                    /
7               hop5                    /
```

Now configure IP addresses, enable routing on some jails, and set the static routes.

```
# ifconfig epair0a inet 192.0.2.0/31 up
# jexec hop1 ifconfig epair0b inet 192.0.2.1/31 up
# jexec hop1 ifconfig epair1a inet 192.0.2.2/31 up
# jexec hop2 ifconfig epair1b inet 192.0.2.3/31 up
# jexec hop2 ifconfig epair2a inet 192.0.2.4/31 up
# jexec hop3 ifconfig epair2b inet 192.0.2.5/31 up
# jexec hop3 ifconfig epair3a inet 192.0.2.6/31 up
# jexec hop4 ifconfig epair3b inet 192.0.2.7/31 up
# jexec hop4 ifconfig epair4a inet 192.0.2.8/31 up
# jexec hop5 ifconfig epair4b inet 192.0.2.9/31 up
# for i in $(jot 4); do jexec hop$i sysctl net.inet.ip.forwarding=1; done
net.inet.ip.forwarding: 0 -> 1
net.inet.ip.forwarding: 0 -> 1
net.inet.ip.forwarding: 0 -> 1
net.inet.ip.forwarding: 0 -> 1
# route add 192.0.2.0/24 192.0.2.1
add net 192.0.2.0: gateway 192.0.2.1
# jexec hop1 route add default 192.0.2.3
add net default: gateway 192.0.2.3
# jexec hop2 route add default 192.0.2.5
add net default: gateway 192.0.2.5
# jexec hop2 route add 192.0.2.0/31 192.0.2.2
add net 192.0.2.0: gateway 192.0.2.2
# jexec hop3 route add default 192.0.2.4
add net default: gateway 192.0.2.4
# jexec hop3 route add 192.0.2.8/31 192.0.2.7
add net 192.0.2.8: gateway 192.0.2.7
# jexec hop4 route add default 192.0.2.6
add net default: gateway 192.0.2.6
# jexec hop5 route add default 192.0.2.8
add net default: gateway 192.0.2.8
```

Test your setup by pinging the fifth jail from the host network stack.

```
# ping -c 2 192.0.2.9
PING 192.0.2.9 (192.0.2.9): 56 data bytes
64 bytes from 192.0.2.9: icmp_seq=0 ttl=60 time=0.265 ms
64 bytes from 192.0.2.9: icmp_seq=1 ttl=60 time=0.482 ms

--- 192.0.2.9 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.265/0.373/0.482/0.108 ms
```

```
# traceroute -n 192.0.2.9
traceroute to 192.0.2.9 (192.0.2.9), 64 hops max, 40 byte packets
 1  192.0.2.1  0.060 ms  0.243 ms  0.244 ms
 2  192.0.2.3  0.180 ms  0.202 ms  0.263 ms
 3  192.0.2.5  0.050 ms  0.159 ms  0.205 ms
 4  192.0.2.7  0.194 ms  0.197 ms  0.191 ms
 5  192.0.2.9  0.261 ms  0.201 ms  0.188 ms
```

Optionally, add a little more fun to this setup (with a big IPv6 range it should be easy to automate a text-to-traceroute script populating DNS configuration file).

```
# cat >> /etc/hosts <<EOF
? 192.0.2.1 once.upon.a.time.in.the.middle
? 192.0.2.3 of.winter.when.the.flakes.of
? 192.0.2.5 snow.were.failling.like
? 192.0.2.7 feathers.from.the.sky
? 192.0.2.9 a.queen.sat.at.a.window.sewing
? EOF
# traceroute 192.0.2.9
traceroute to 192.0.2.9 (192.0.2.9), 64 hops max, 40 byte packets
 1  once.upon.a.time.in.the.middle (192.0.2.1) 0.237 ms  0.508 ms  0.540 ms
 2  of.winter.when.the.flakes.of (192.0.2.3) 0.489 ms  0.361 ms  0.373 ms
 3  snow.were.failling.like (192.0.2.5) 0.343 ms  0.337 ms  0.285 ms
 4  feathers.from.the.sky (192.0.2.7) 0.255 ms  0.296 ms  0.271 ms
 5  a.queen.sat.at.a.window.sewing (192.0.2.9) 0.328 ms  0.271 ms  0.242 ms
```

Going Further

Connecting Jails with the Outside World

Multiple choice here:

1. With SR-IOV compliant NIC, generate multiple Virtual NIC and assign them to the jails.
2. Virtual-Interface (drivers specific)
3. Using VLAN and assigning VLAN interfaces to each jail, the limitation is one jail per VLAN maximum per Ethernet port.
4. Using if_bridge interface (and it's possible to mix with VLAN, too) is the easiest setup but there is some performance penalty when using if_bridge.

SR-IOV

This feature, initially designed for virtual machine use, creates multiple Virtual Function (VF = Virtual NIC in our case). And by using the default non-passthrough mode, it will present multiple virtual NIC to the host, each of which could be attached to vnet-jail.

Here is an example using two Chelsios interfaces (cxl0 and cxl1) to create 10 VF for each.

```
# sysrc iovctl_files="/etc/iovctl.cxl0.conf /etc/iovctl.cxl1.conf"
# sysrc kld_list+=if_cxgbev
# cat > /etc/iovctl.cxl0.conf <<EOF
? PF {
? device : "cxl0";
? num_vfs : 10;
```

```
? }
? EOF
# cat > /etc/iovctl.cxl1.conf <<EOF
? PF {
? device : "cxl1";
? num_vfs : 10;
? }
? EOF
# iovctl -C -f /etc/iovctl.cxl0.conf
# iovctl -C -f /etc/iovctl.cxl1.conf
# kldload if_cxgbev
# tail /var/log/messages
(...) kernel: t5vf18: <Chelsio T540-CR VF> at device 0.41 on pci4
(...) kernel: cxlv18: <port 0> on t5vf18
(...) kernel: cxlv18: Ethernet address: 06:44:2e:e5:90:18
(...) kernel: cxlv18: 2 txq, 1 rxq (NIC)
(...) kernel: t5vf18: 1 ports, 2 MSI-X interrupts, 4 eq, 2 iq
(...) kernel: t5vf19: <Chelsio T540-CR VF> at device 0.45 on pci4
(...) kernel: cxlv19: <port 0> on t5vf19
(...) kernel: cxlv19: Ethernet address: 06:44:2e:e5:90:19
(...) kernel: cxlv19: 2 txq, 1 rxq (NIC)
(...) kernel: t5vf19: 1 ports, 2 MSI-X interrupts, 4 eq, 2 iq
# ifconfig -l
cxl0 cxl1 igb0 lo0 cxlv0 cxlv1 cxlv2 cxlv3 cxlv4 cxlv5 cxlv6 cxlv7 cxlv8 cxlv9 cxlv10
cxlv11 cxlv12 cxlv13 cxlv14 cxlv15 cxlv16 cxlv17 cxlv18 cxlv19
```

Now we can keep cxl0 (=physical interface) for the host and all cxlvX interfaces can be assigned to a different jvnet-jail.

Notice that:

1. some NIC (like Intel) need more parameters (allow-promisc, allow-set-mac, mac-anti-spoof) to allow specific usage like CARP with the VF.
2. A FreeBSD 12 with Intel ix(4) drivers couldn't attach drivers to these VF.

```
ixv0: <Intel(R) PRO/10GbE Virtual Function Network Driver> at device 0.128 on pci4
ixv0: ...reset_hw() failure: Reset Failed!
ixv0: IFDI_ATTACH_PRE failed 5
device_attach: ixv0 attach returned 5
```

Virtual-Interface (driver specific)

Chelsio driver supports another mode called Virtual Interface; here is an example asking to create four VI per port.

```
# echo hw.cxgbe.num_vis=\"4\" >> /boot/loader.conf
```

After a reboot the new interfaces (vcxlX) will be available and show in dmesg as:

```
vcxl0: <port 0 vi 1> on cxl0
vcxl0: Ethernet address: 00:07:43:2e:e5:91
vcxl0: netmap queues/slots: TX 2/1023, RX 2/1024
vcxl0: 1 txq, 1 rxq (NIC); 2 txq, 2 rxq (netmap)
vcxl1: <port 0 vi 2> on cxl0
vcxl1: Ethernet address: 00:07:43:2e:e5:92
vcxl1: netmap queues/slots: TX 2/1023, RX 2/1024
vcxl1: 1 txq, 1 rxq (NIC); 2 txq, 2 rxq (netmap)
(...)
```

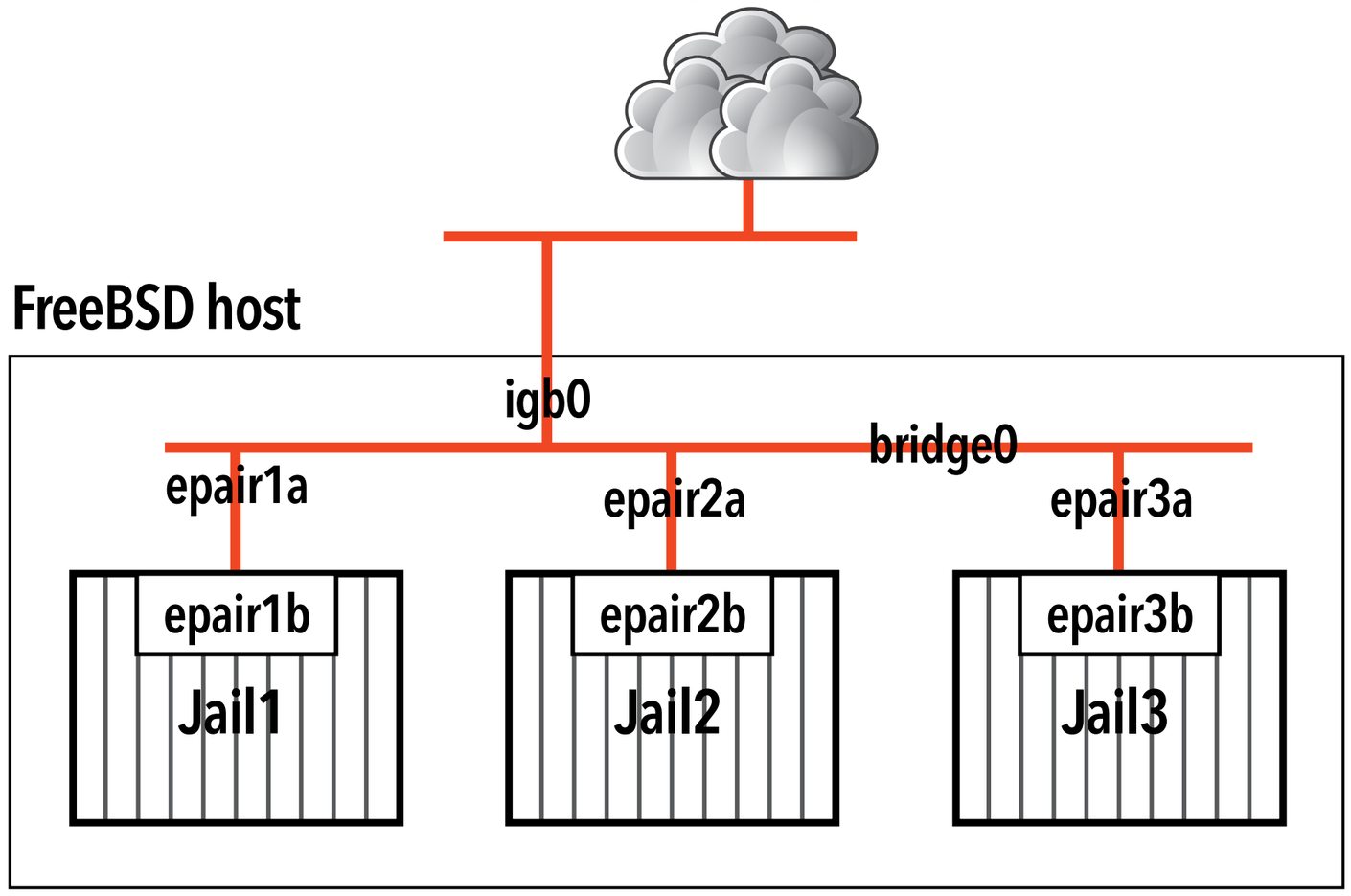
Now you can assign these Virtual interfaces (vxclX) to the vnet-jail.

VLAN
Without a NIC supporting SR-IOV or Virtual-Interface features, one other possibility is to create multiple VLAN and assign the VLAN interface to the vnet-jail. The restriction is that VLAN ID are unique per interface, so if two vnet-jails need to be assigned a vlan sub-interface in the same VLAN, you need to use two physical interfaces.

```
# ifconfig igb0.6 create vlan 6 vlandev igb0 up
# ifconfig igb0.7 create vlan 7 vlandev igb0 up
# ifconfig igb1.6 create vlan 6 vlandev igb1 up
```

In this example, new interfaces igb0.6, igb0.7, and igb1.6 are available for the vnet-jails.

Bridge + epair
To remove the restriction of unique VLAN ID per physical interface, there is still the classical approach, but with some performance impact using bridge and epair setup.



This small diagram is generated with these commands:

```
# ifconfig bridge create up
bridge0
# for i in $(jot 3); do ifconfig epair$i create up; done
```

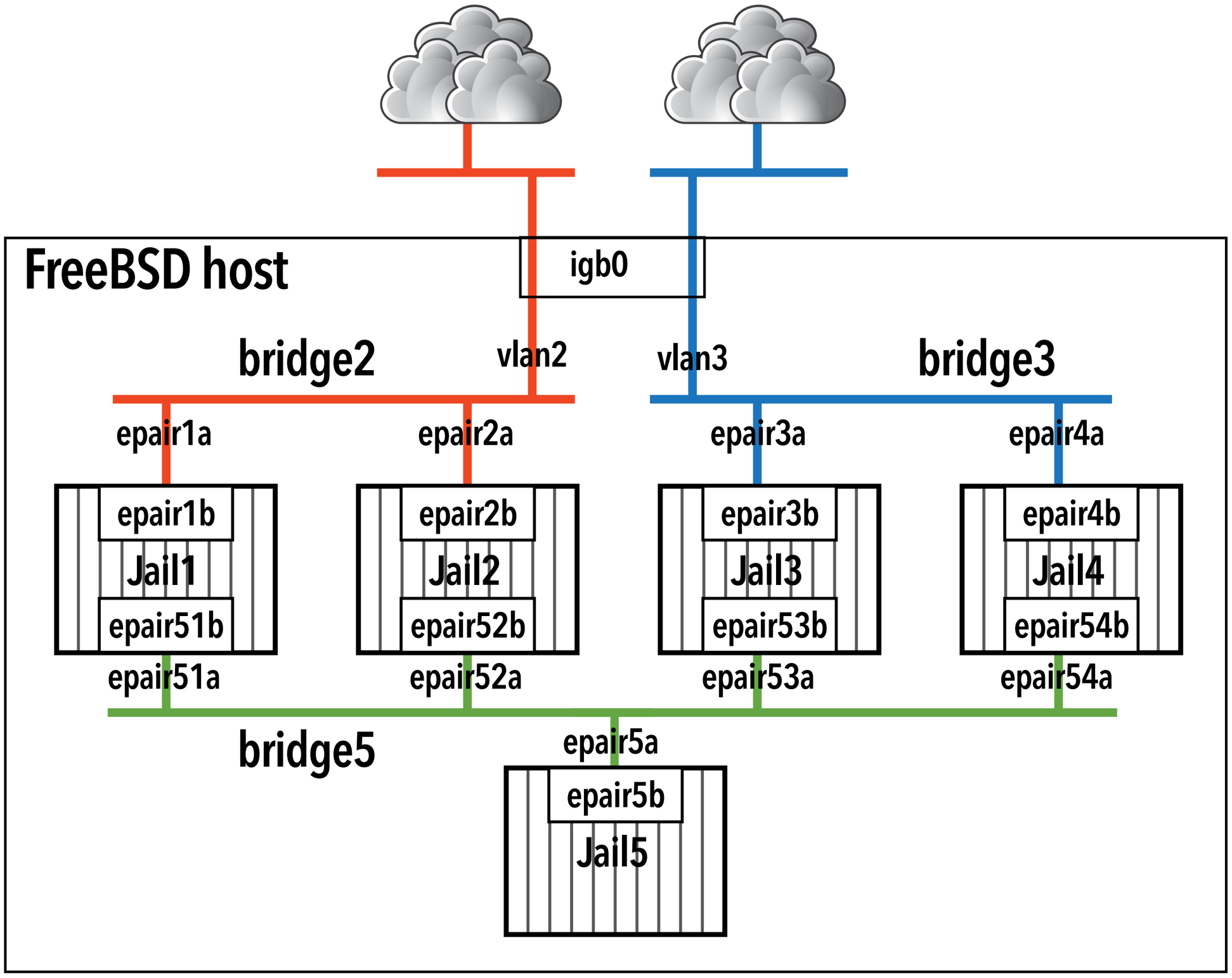
```
epair1a
epair2a
epair3a
# ifconfig bridge0 inet 192.0.2.254/24 addm igb1 addm epair1a addm epair2a \
addm epair3a
# for i in $(jot 3); do jail -c name=jail$i host.hostname=jail$i persist vnet \
vnet.interface=epair${i}b; jexec jail$i ifconfig epair${i}b inet \
192.0.2.${i}/24 up; jexec jail$i ifconfig epair${i}b inet; done
epair1b: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
        options=8<VLAN_MTU>
        inet 192.0.2.1 netmask 0xffffff00 broadcast 192.0.2.255
epair2b: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
        options=8<VLAN_MTU>
        inet 192.0.2.2 netmask 0xffffff00 broadcast 192.0.2.255
epair3b: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
        options=8<VLAN_MTU>
        inet 192.0.2.3 netmask 0xffffff00 broadcast 192.0.2.255
# ping -c 2 192.0.2.1
PING 192.0.2.1 (192.0.2.1): 56 data bytes
64 bytes from 192.0.2.1: icmp_seq=0 ttl=64 time=0.158 ms
64 bytes from 192.0.2.1: icmp_seq=1 ttl=64 time=0.103 ms

--- 192.0.2.1 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.103/0.130/0.158/0.027 ms
# ping -c 2 192.0.2.2
PING 192.0.2.2 (192.0.2.2): 56 data bytes
64 bytes from 192.0.2.2: icmp_seq=0 ttl=64 time=0.189 ms
64 bytes from 192.0.2.2: icmp_seq=1 ttl=64 time=0.104 ms

--- 192.0.2.2 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.104/0.146/0.189/0.042 ms
# ping -c 2 192.0.2.3
PING 192.0.2.3 (192.0.2.3): 56 data bytes
64 bytes from 192.0.2.3: icmp_seq=0 ttl=64 time=0.201 ms
64 bytes from 192.0.2.3: icmp_seq=1 ttl=64 time=0.091 ms

--- 192.0.2.3 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.091/0.146/0.201/0.055 ms
```

And to clean up this example.

```
# for i in $(jot 3); do jail -R jail$i; ifconfig epair${i}a destroy; done
# ifconfig bridge0 destroy
```

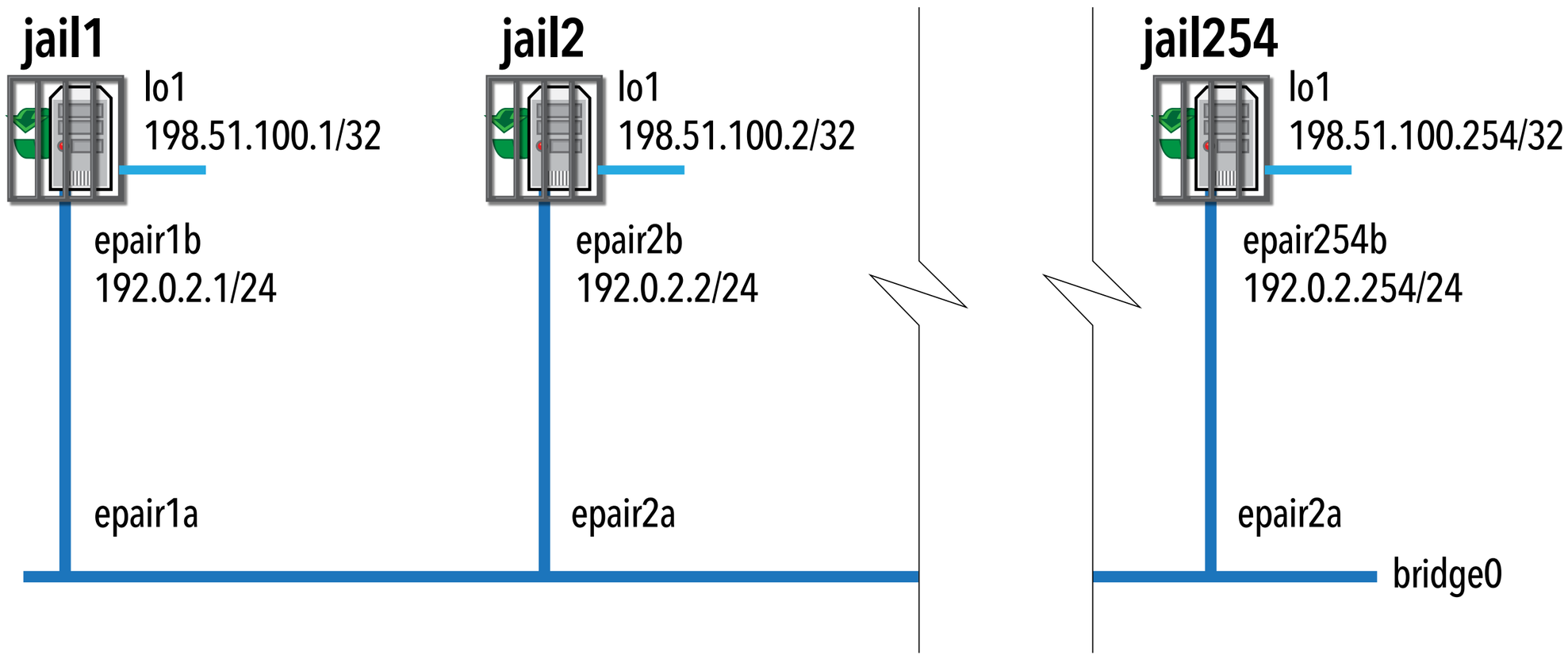
Final Exercise

Now you should be able to set up this kind of lab on your own.



How Light Is a Jail Compared to a VM?

Let's use a small device like PC Engines APU2 (4 core AMD GX-412TC SOC 1Ghz, 4GB of RAM, 16Gb flash). On this device, how many jails executing a real process (bird using OSPF to announce each jail's loopback) can we start?



This small shell script is used to start 480 jails with a bird OSPF tuned to use large numbers of neighbors on a shared link:

- MTU increased to 9000, allowing large numbers of neighbors (you can only have about 350 maximum OSPF neighbors with the default 1500 bytes MTU)
- Hello and dead interval increased to reduce multicast storm on the bridge interface

```
#!/bin/sh
set -eu
dec2dot () {
    # $1 is a decimal number
    # output is pointed decimal (IP address format)
    printf '%d.%d.%d.%d\n' $(printf "%x\n" $1 | sed 's/../0x& /g')
}
# Need to increase some network value a little bit
# to avoid "No buffer space available" messages
# maximum number of mbuf clusters allowed
sysctl kern.ipc.nmbclusters=1000000
sysctl net.inet.raw.maxdgram=16384
sysctl net.inet.raw.recvspace=16384
# Start addressing shared LAN at 192.0.2.0 (in decimal to easily increment it)
ipepairbase=3221225984
# start addressing loopbacks at 198.51.100.0
iplobase=3325256704
ifconfig bridge create name vnetdemobridge mtu 9000 up
for i in $(jot 480); do
    ifconfig epair$i create mtu 9000 up
    ifconfig vnetdemobridge addm epair${i}a edge epair${i}a
    jail -c name=jail$i host.hostname=jail$i persist \
         vnet vnet.interface=epair${i}b
    ipdot=$( dec2dot $(( iplobase + i)) )
    jexec jail$i ifconfig lo1 create inet ${ipdot}/32 up
    ipdot=$( dec2dot $(( ipepairbase + i)) )
    jexec jail$i ifconfig epair${i}b inet ${ipdot}/20 mtu 9000 up
    cat > /tmp/bird.${i}.conf <<EOF
protocol device {}
protocol kernel { ipv4 { export all; }; }
protocol ospf {
  area 0 {
    interface "epair${i}b" {
      hello 60;
      dead 240;
    };
    interface "lo1" {
      stub yes;
    };
  };
}
EOF
```

```
    jexec jail$i bird -c /tmp/bird.$i.conf -P /tmp/bird.$i.pid \
          -s /tmp/bird.$i.ctl -g birdvty
done
```

Install net/bird2, execute this script, and after the OSPF DR/BDR election and database synchronization, network traffic on the bridge interface should still be quite high with only the OSPF keep-alives on the bridge interface:

```
# netstat -ihw 1 -I vnetdemobridge
            input vnetdemobridge              output
   packets  errs idrops      bytes    packets  errs      bytes colls
       490     0     0       826K       29k     0       7.0M     0
       981     0     0       1.7M       65k     0        22M     0
      1.5k     0     0       3.3M       92k     0        25M     0
       596     0     0       337K      102k     0        35M     0
       732     0     0       479K      100k     0        33M     0
```

After a few minutes, check number of neighbors detected (should be 479). DR/BDR election should have chosen jail479 as BDR and jail480 as DR and number of learned routes.

```
# birdcl -s /tmp/bird.1.ctl show ospf
BIRD 2.0.6 ready.
ospf1:
RFC1583 compatibility: disabled
Stub router: No
RT scheduler tick: 1
Number of areas: 1
Number of LSAs in DB:   481
        Area: 0.0.0.0 (0) [BACKBONE]
                Stub:   No
                NSSA:   No
                Transit:        No
                Number of interfaces:   2
                Number of neighbors:    479
                Number of adjacent neighbors:   2

# jexec jail1 netstat -4rn | grep UGH1 | wc -l
     479
```

The current system limit of this test is due to 4GB of RAM consumed by all the bird processes.

```
last pid: 16459;  load averages: 34.76, 37.35, 28.18
up 0+00:36:15  08:36:46
497 processes: 1 running, 496 sleeping
CPU: 14.4% user,  0.0% nice,  6.3% system, 10.2% interrupt, 69.2% idle
Mem: 1177M Active, 454M Inact, 2816K Laundry, 1891M Wired, 17M Buf, 395M Free
```

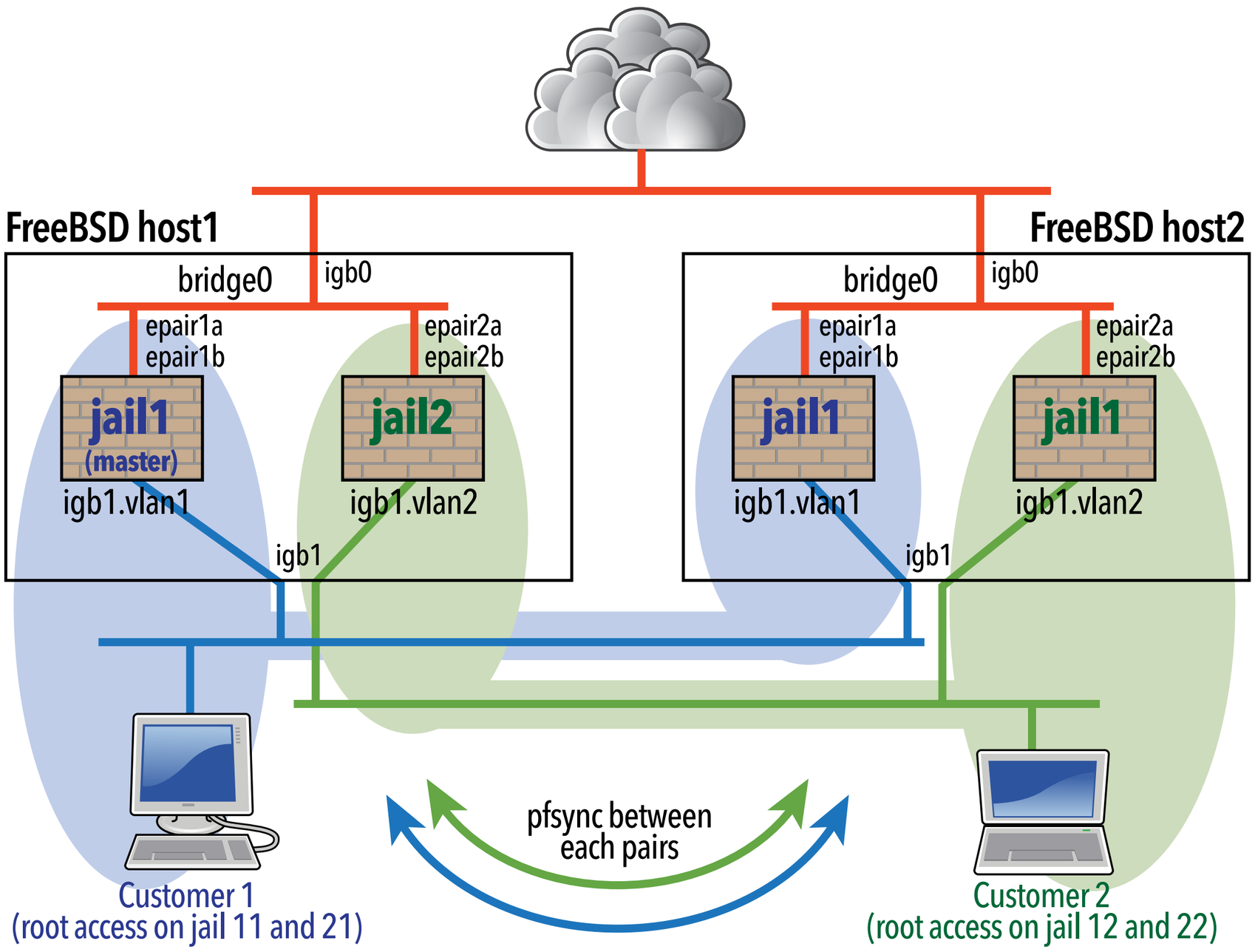
```
  PID USERNAME    THR PRI NICE   SIZE    RES STATE    C   TIME    WCPU COMMAND
13529 root          1  20    0    32M    20M select   1   0:45   2.04% bird
13553 root          1  20    0    29M    17M select   3   0:41   0.71% bird
16459 root          1  20    0    14M  3568K CPU1     1   0:00   0.62% top
13512 root          1  20    0    20M  7372K select   2   0:03   0.39% bird
 8003 root          1  20    0    20M  7316K select   0   0:03   0.38% bird
 7913 root          1  20    0    20M  7260K select   0   0:03   0.38% bird
13466 root          1  20    0    20M  7172K select   0   0:03   0.34% bird
 7887 root          1  20    0    20M  7288K select   1   0:03   0.33% bird
 7832 root          1  20    0    20M  7260K select   2   0:03   0.33% bird
```

Here is the script to delete/clean up all the jails, but you should reboot because your system will panic during this cleanup:

```
#!/bin/sh
set -eu
for i in $(jot 480); do
    echo Deleting jail$i
    jail -R jail$i
    ifconfig epair${i}a destroy
    rm /tmp/bird.$i.*
done
ifconfig vnetdemobridge destroy
```

Firewalls

pf and ipfw are vnet compliant, which allows the building of a multi-tenant firewall in an HA scenario like the one presented here:



This setup is more complex because it needs to enable specific kernel features to be useable as a jail. (A detailed example of such setup could be presented in a future article.) Meanwhile, this setup is explained as a BSD Router Project example here (using some helper scripts to configure a jail that hides complexity):
https://bsdrp.net/documentation/examples/multi-tenant_ha_pf_firewalls

---

OLIVIER COCHARD-LABBÉ discovered FreeBSD in 2005 by customizing m0n0wall to create FreeNAS. As a network engineer, he created the BSD Router Project in 2009 and since then has been endeavoring to bench the FreeBSD network stack. He received his port commit bit in 2016 and is currently a software developer in tests at Netflix.

FreeBSD®

The FreeBSD Project is looking for

- Programmers
- Testers
- Researchers
- Tech writers
- Anyone who wants to get involved

Find out more by

Checking out our website
freebsd.org/projects/newbies.html

Downloading the Software
freebsd.org/where.html

We're a welcoming community looking for people like you to help continue developing this robust operating system. Join us!

Already involved?

Don't forget to check out the latest grant opportunities at
freebsdfoundation.org

Help Create the Future. Join the FreeBSD Project!

FreeBSD is internationally recognized as an innovative leader in providing a high-performance, secure, and stable operating system.

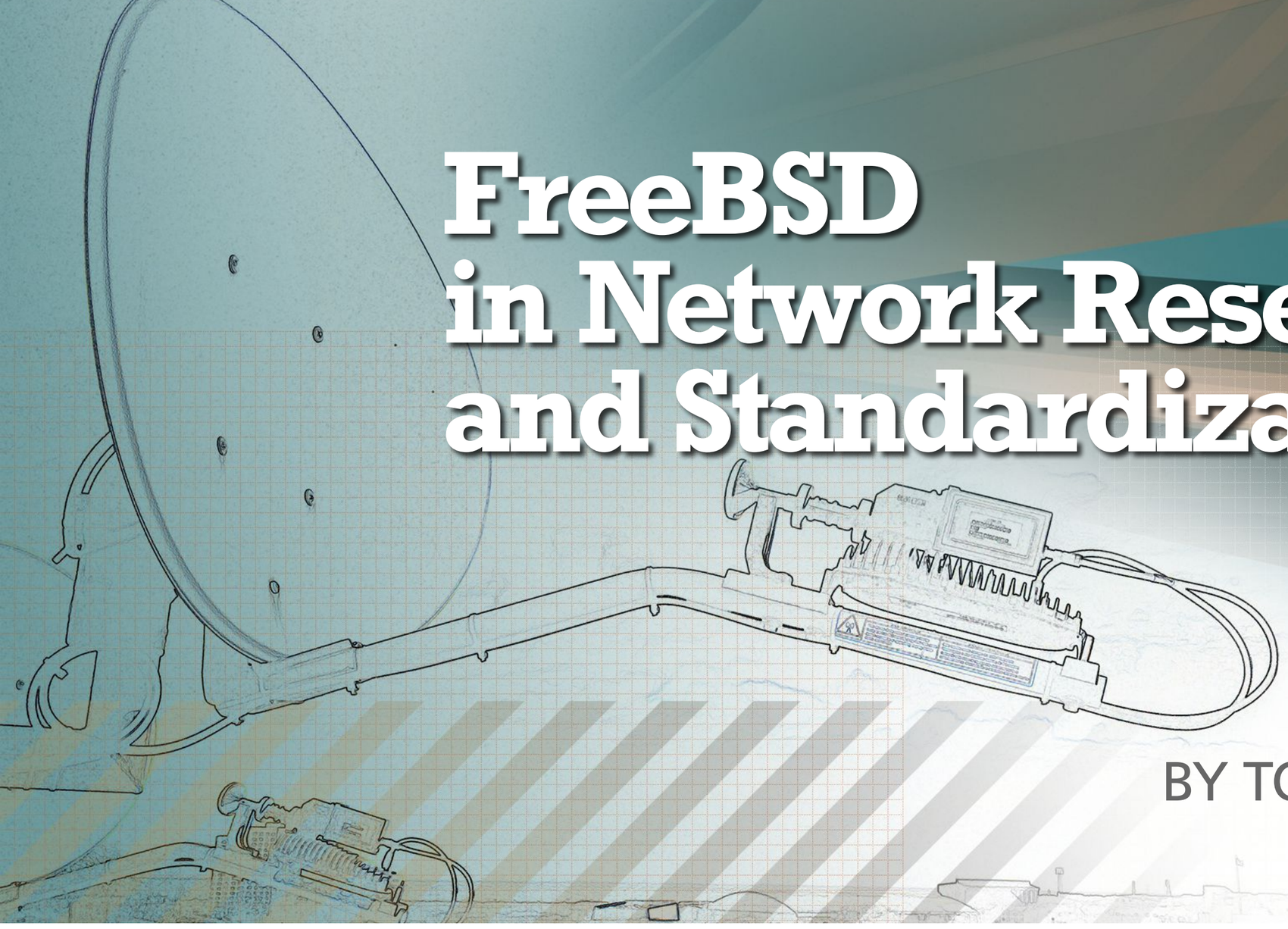
Not only is FreeBSD easy to install, but it runs a huge number of applications, offers powerful solutions, and cutting edge features. The best part? It's FREE of charge and comes with full source code.

Did you know that working with a mature, open source project is an excellent way to gain new skills, network with other professionals, and differentiate yourself in a competitive job market? Don't miss this opportunity to work with a diverse and committed community bringing about a better world powered by FreeBSD.

The FreeBSD Community is proudly supported by

FreeBSD FOUNDATION

FreeBSD in Network Research and Standardization



BY TOM JONES

Our research looks at Internet protocols and how they can be improved to make the Internet better. We perform measurements on the network to get a picture of how things really are deployed, and from these measurements we develop new mechanisms, enhancements, and improvements to Internet protocols. Our work outputs into and advises Internet standards with the Internet Engineering Task Force (IETF), and we publish our work publicly so that the entire Internet community can benefit.

Our research group has been involved with the IETF for over three decades, contributing to a wide range of standards. Two of our efforts in recent years have been trying to adapt the Internet to support large packet sizes and making sure that new protocols work well in satellite networks.

FreeBSD is a vehicle for a lot of our work. By using FreeBSD both for research test beds and as a target for new features, we are able to complete work that is on par with what can be performed by a much larger organization.

FreeBSD has become an important part of our research, and in the last few years, we have used FreeBSD in three distinct ways:

- FreeBSD with ZFS makes a core component of the storage network for our research group.
- FreeBSD and dummynet are used as a core component in network experiments we perform.
- The FreeBSD network stack is a high-performance venue for research on new protocol features.

Our research group is very lucky to have our own network within the University of Aberdeen. This is becoming more and more rare as the IT service departments have taken over running computers from computer science and engineering. We run all of the infrastructure in our labs and for our test beds, using the university as an upstream provider to access the UK research network JANET.

With our own separate network, we are able to host and run experiments from IP address space that we manage. This gives us a great amount of flexibility when putting together an experiment, and we are able to bring up a test network in just the time it takes to configure the machines. In many other institutions, any network operation requires months of liaison and interaction with an IT department in which a very strong case needs to be made before anything out of the ordinary is considered. We maintain a strong relationship with network operations at the university and they are key to helping us maintain our independence.

From our network, we are able to perform large-scale measurements of the Internet, and these measurements feed into protocol design work and standardization. Without our vantage point and a friendly upstream, we would have to do scans of the entire Internet from cloud providers. The complexities of setting up a measurement from a cloud provider is hard to imagine, whereas we are able to spin up an experiment to answer a question we have in short order.

Internet Standardization in the IETF

The Internet Engineering Task Force is the standards body that defines the protocols with which the Internet is built. The IETF differs from other standards bodies (such as ISO or IEEE) in many ways, and the deliberately open process and volunteer core really stand out.

The IETF process is open to anyone who can subscribe to a mailing list—the only barrier to entry to take part is taking part. This is quite dramatically different from other organizations, as this open model allows universities like ours to be part of enhancing the Internet on the same level as billion-dollar companies and countries.

The standards the IETF defines go through an intense authorship process. They are started as Internet drafts written or curated by a group of volunteer authors. These documents may be adopted by a working group where they are reviewed and commented on, and the ideas within are improved through a process that can take a couple of months or several years. Once the idea has settled and has been thoroughly reviewed, the document may be published as an RFC with its own RFC number.

Rough consensus and running code is one of the core precepts of the IETF. Working implementations is one of the metrics used to evaluate a new idea, with the IETF preferring multiple interoperable implementations of a draft while it is being developed toward becoming an RFC. There really are only two open-source operating systems that have a mature, high-quality, high-performance network stack, FreeBSD and Linux.

It can be very helpful for a draft to see implementation so that it can be tested in the environment in which it will be deployed and so that the actual standards language that defines the protocol can be carefully checked.

We have worked on implementations of many different protocols in both Linux and FreeBSD at all stages of their life cycles, including documents we have written and documents from others. Some of these ideas continue through to becoming RFCs while others are not proven in the IETF process and are put to the side.

We have worked on new transport protocols (DCCP, UDP Lite, and UDP Options), enhancements to existing protocols such as TCP (TCP ABE RFC 8511, NewCWV RFC 7661), and new network layer mechanisms (Hop by Hop MTU) as well as developed new protocol mechanisms (Datagram PLPMTUD) and given advice on defaults to help in satellite networks.

We have also worked on documenting and improving the Sockets API (RFC8304) and have been involved in the TAPS working group to create a next generation transport framework. We are working on improvements to the existing socket API that arise from the development of transport protocols that run in user space or on top of UDP.

FreeBSD and Dummynet to Emulate Satellites

Our research group has a long history working with satellite networks. The added delay from a round-trip out to geostationary orbit can have large effects on how a transport protocol performs. Recently, we have been working with the European Space Agency to look at how well the QUIC transport protocol works in satellite networks.

QUIC is a new UDP-based transport protocol designed to replace http/2. The protocol was originally developed by Google and has been taken on by the IETF as a work item. It has taken

a lot of hard work, but the working group is now approaching the release of QUICv1. QUICv1 has been designed with a new HTTP layer, HTTP3, which is to replace HTTP2 in the web, using a protocol that is comprehensively encrypted. Strong encryption and authentication form the roots of QUIC, to the point that most protocol headers are obscured from the network.

Satellite operators and providers are worried about the impact of the complete encryption. Part of the reason why TCP works well over a satellite network is that a proxy normally runs on the satellite terminal that enhances TCP. One of the ways these proxies make TCP usable with the high and variable satellite delay is by splitting the connection at the home terminal and at the satellite gateway. This split is transparent TCP, but it has a very large beneficial effort on TCP's performance.

Our KA Band satellite dishes that connect to the HYLAS 1 satellite looking over the noted Sir Duncan Rice library building at the University of Aberdeen.

QUIC specifically sets out to make such proxies impossible to implement. With performance enhancing proxies no longer available, QUIC has to be evaluated on its own satellite network cases. The Internet community and satellite operators need to work together to make their network practical for future deployments.

Our test bed network has a satellite link similar to the sort you might get as a DSL replacement for your remote cabin in the mountains. We have in the past used this link to look at how the Internet is developing and changing for people in rural Scotland. We have a limited capacity on this link each day. For us to be able to do repeated experiments on satellite paths, we had to create an emulated link that matched the real satellite as best as possible. Having an emulated link to confirm tests also helps make our work reproducible by others who are not fortunate enough to have a cool satellite to play with.

There are two options for doing high-quality network emulation. Linux offers network emulation module (netem), and FreeBSD has the dummynet framework that works with IPFW. We performed a series of experiments on the link to get a model of its typical characteristics and got numbers for the forward (downstream) and return (downstream) link capacity and the delay of the network. The geostationary satellite system that we use has a typical delay of 600ms, which is huge compared to what you would see on customer DSL line.

We like to be thorough when creating a test bed, and we configure both FreeBSD dummynet and Linux netem test beds. Clock and timing issues make building an emulation network in virtual machines impractical, which is a shame, but we are unsure if there is any way out. Network namespaces in Linux do seem to get around a lot of the timing issues while being able to run many nodes on one host. Unfortunately, dummynet does not work in vnet Jails in FreeBSD yet.

With the requirements for discrete hosts, we tend to use PC ENGINES APU 2 boards, which are supported well in FreeBSD and Linux and are cheap enough that we have a small cluster of them for network experiments.

FreeBSD shines over Linux when it comes to configuring the network for emulation. We are unable to easily get Linux performance in line with what we need due to the design of the netem network emulator. We have found that packets were able to bypass the delay and capacity limits we put on them. It is hard to be sure of the results when occasionally a packet is able to cross the network 100 times faster than normal. The difficultly with configuring netem has hit other people recently, and we have seen other groups default to using dummynet and FreeBSD for satellite network emulation.

FreeBSD Enhancements

Our FreeBSD kernel work started in 2014. At that time, we were working on standardizing a modification to TCP called NewCWV. NewCWV defines new mechanisms for how a TCP acts when traffic is sent in bursts such as when you are watching streaming video.

We had an implementation of NewCWV for Linux that was developed while we were still working on the protocol mechanisms. We made an attempt to upstream this code into Linux, but the Linux networking community is quite closed, and we got a cold response. The IETF likes running code, and the more running code the better. We had a window of time before a project could advance, and I suggested that I port the code we had from Linux to FreeBSD.

Getting code accepted by an open-source project really is as difficult or more difficult than the implementation. An offer from a passerby might be a bug fix or a great new feature, but all code needs to be maintained, and the cost of taking something on always has to be weighed. When the offered code is an experimental feature that touches a core network path, it is much harder to know if it is a good idea to take the code.

Running code is very important in the IETF process—an idea is much more likely to get truncation if someone has put in the effort to do an implementation. When an idea is in its early stages, it is very easy to have a large impact by prototyping. The accessibility of FreeBSD and the welcoming nature of the community mean that we can spend some time playing with an implementation in FreeBSD. If the idea doesn't pan out (maybe the approach is not quite right, or it isn't the right time), we can give feedback to the authors of the idea. This allows us to have a very active review part in the IETF process.

This can be seen in the 6MAN working group in the IETF. The 6MAN working group maintains IPv6, and it is where extensions to the protocol are developed. Recently there has been an interest in 6MAN to fix path MTU discovery (something that is problematic for all network operators and sysadmins). In 2018 and 2019, there were six or seven different ideas to try different methods to make path MTU probes. We were able to prototype some of these and find the implementation difficulties. This work is still in development, but we are safe in investing a small amount of time to try something out, because if the idea does mature, we know it will get a fair review by the FreeBSD community.

Over the past three years, we, along with Muenster University of Applied Sciences, have been working on a new method to use larger packet sizes on the Internet. Our new algorithm Datagram Packetization Layer Path MTU Discovery (my favorite standards-based tongue twister) is designed to work with datagram protocol whether in user space or in the kernel. The standards part of this algorithm is approaching maturity (entering the final stages of working group process at the start of 2020), and we are proud to say that there are many implementations, including one for FreeBSD's SCTP procotol stack by Julius Flohr while he was visiting our group at the end of 2019.

The FreeBSD community is much easier to understand and approach when you have something new you would like to share. Once you find the right set of people, the FreeBSD community is very friendly and there are developers that will help you progress a patch or a new idea into something great.

The Future

FreeBSD has a long lineage in the research community as both a target and research vehicle. I believe that FreeBSD will continue to be a target for transport and network protocol advancement.

As a platform for research, I do think we are falling behind the zeitgeist. The devops bandwagon rolled in with Docker, and it has had a big impact in the way that experiments and measurement campaigns can be put together. We partnered in a research test bed as part of H2020 Fire project that used docker images as a core component for deploying experiments.

Jails and vnet might have been both nicer and easier to use for the platform, but the mind share is with Docker, and it is hard to justify building out a new system. There are a number of projects to make something more Docker-like for FreeBSD, and we hope these can develop to the point where they can provide the same feature set that is available in the Linux container ecosystem.

We aim to make our experiments reproducible, and there is a lot to be said for devops style pipelines as a method for deploying and running an experiment. Docker has caused a large number of packages and APIs to be available that make programmable tooling possible. While it is possible to hack together a similar system, too often we are required to shell out to manage an interface or configure a firewall.

Dummynet has been around for a very long time, and it is core to our usage of FreeBSD in test bed networks. Dummynet has not seen active development for several years, and while there have been recent changes to allow greater than 2Gb/s rules, there has not been the attention required for it to support traffic shaping and emulation at 10 and 100 gigabit speeds. Dummynet still works well for our satellite use cases as the capacity required is quite low. For other groups that might need very high-capacity networks, however, I am not sure that dummynet will be available as a choice.

Conclusion

For us, FreeBSD is both part of our lab infrastructure and a target for science. Using FreeBSD as a target for our work has given us a high familiarity with it, and that continued exposure has made us want to use FreeBSD as a platform for experiments. FreeBSD's base makes it easy for us to build test beds and it actively helps with our experimentation, but there is still more that could be done to make it the perfect platform for science.

The FreeBSD community is welcoming and encouraging of new work, and this fact makes it reasonable for us to propose taking a new idea into an operating system. Without this community, it is likely that we wouldn't experiment with implementing new ideas from the IETF. FreeBSD is an important part of our work, and we hope to keep using it for many years to come.

TOM JONES is a researcher in the Electronics Research Group in the School of Engineering at the University of Aberdeen in the North East of Scotland. His work looks at Internet transport protocols and their performance with a focus on satellite networks. He has worked on standardization in the IETF trying to define next generation APIs and algorithms to help the Internet grow.

Support FreeBSD®

Donate to the Foundation!

You already know that FreeBSD is an internationally recognized leader in providing a high-performance, secure, and stable operating system. It's because of you. Your donations have a direct impact on the Project.

Please consider making a gift to support FreeBSD for the coming year. It's only with your help that we can continue and increase our support to make FreeBSD the high-performance, secure, and reliable OS you know and love!

Your investment will help:

- Funding Projects to Advance FreeBSD
- Increasing Our FreeBSD Advocacy and Marketing Efforts
- Providing Additional Conference Resources and Travel Grants
- Continued Development of the FreeBSD Journal
- Protecting FreeBSD IP and Providing Legal Support to the Project
- Purchasing Hardware to Build and Improve FreeBSD Project Infrastructure

Making a donation is quick and easy.

freebsdfoundation.org/donate

FreeBSD™
FOUNDATION

AN • INTERVIEW • WITH

Trenton Schulz

by Allan Jude and Benedict Reuschling

The *FreeBSD Journal* Editorial Board suggested that some of the outstanding interviews from the BSD Now series might be of interest to readers as they reflect the state of technology when the interview took place. Here, we've transcribed and excerpted from an October 2019 interview with Trenton Schulz by Allan Jude and Benedict Reuschling.

BENEDICT REUSCHLING: We have a special interview with Trenton Schulz about his early days with FreeBSD, the Robot Operating System (ROS), Qt, and more.

Hello, I'm your host, Benedict Reuschling.

ALLAN JUDE: And I'm Allan Jude.

BENEDICT: Trenton Schulz will talk with us about Robot OS on FreeBSD. First, welcome to the show, Trenton. Can you tell us a little bit about yourself and how you got started with BSD?

TRENTON SCHULZ: Sure, well, thank you very much, Benedict. I work as a researcher at the Norwegian Computer Center. Previously, I was a PhD student and a software engineer. I got started in BSD when I was working on my bachelor's degree back in Minnesota, in the late 1990s, and it was a game night at one of the computer labs when everybody was allowed to bring in and play computer games on the lab computers.

Except there were a couple of people in the back of the room who had brought in their own computers. And they had done that so they could access the network, and they were installing this weird DOS thing on their computers, or at least that's what it looked like to me. And so, I asked, "What are you doing?"

And they said, "Well, we're installing FreeBSD." And I responded, "What is FreeBSD?" And they explained, "It's an operating system."

I said, "You mean like Windows, or whatever?" And they said, "Yeah, but it's UNIX," or something like that. And I had heard about UNIX at that point, so I was kind of interested. Within a week or two, I had brought in my computer and installed it. And then there were about three of us at the university who were working with FreeBSD and sending questions back and forth to each other or looking in the mailing lists, and things like that.

I think I installed FreeBSD 2.2.5, and of course, it was interesting just getting the software to work. I remember complaining that the sound didn't work when I wanted to listen to music or something like that. And that meant upgrading to -CURRENT.

[One of those] friends had no problem running build world on my computer and upgrading it to -CURRENT. Of course, then other things didn't work.

I got to learn a little bit about how these operating system things work.

ALLAN: Hmm, tell us how you got your first job related to BSD.

TRENTON: Well, it was sort of tangentially. While I was doing my computer science studies, I was interested in being able to do graphics on computers. And since I was using FreeBSD and the X Windows System, I wanted to do graphics on that. I was looking for toolkits that I could

work with to do that, and since at that time at that college they were teaching us how to do C++ programming, I found the Qt library. Or "cute" as it's called among all the people who program for it.

I started working on some small Qt programs on my own, and then I kind of rolled them all up into a little package, sent them off to Trolltech, with my resume that I had written in plain text. But they were looking for employees and I got hired. At that time, Norway kind of had a shortage of C++ programmers who weren't in the oil industry.

I ended up coming to Norway, and I got a FreeBSD box when I got to Trolltech. At that time, I was helping out with the Qt/embedded port, which was using the frame buffer. Of course, I was interested in trying to get Qt/Embedded to work with FreeBSD, but I didn't know nearly enough [about device drivers] to do that.

There was a virtual frame buffer that worked on X11 that was just a shared memory so you could actually display things and use the Qt/embedded windowing system [and X at the same time].

And I remember the first weekend I was in Oslo, or the first couple of days, I was at Trolltech fixing the Qt virtual frame buffer to actually work on FreeBSD. In the end, it was just using some Linux-specific semaphore calls and then just changing them over to use the FreeBSD ones. And it worked on both Linux and FreeBSD. I think that was probably the first commit I did to Qt.

I actually shared an office with a guy named Brad, who was also a giant FreeBSD person. He had worked on the Blackbox window manager, which was the forerunner to the Fluxbox window manager.

We were the two FreeBSD people at Trolltech, and the idea was to make sure that Qt was somewhat semi-operable on other operating systems. Or other UNIXes I should say.

BENEDICT: Oh, that's great.

ALLAN: How long were you there?

TRENTON: I was at Trolltech for—well, I guess seven years until they became part of Nokia, and then I stayed on for a year at Nokia as well. At that point I had moved off FreeBSD and was working with Mac.

[The switch to Mac was because] sometimes everything would get out of whack with FreeBSD, and then you would run this portupgrade utility and try to rebuild all your ports, and it was really getting annoying. There was nothing like package or anything that we have these days.

ALLAN: Exactly.

TRENTON: I think it was a point there where I was willing to let a vendor do all that extra maintenance work so I could program. Trolltech had hired me to program on Qt, not maintain my system.

ALLAN: Yes, sort of maintaining an operating system's use of Qt and so on.

TRENTON: I moved over to the Mac for a while then. Then, I finished one of these big projects and I felt that I needed to try something different. So I jumped into research, which is interesting in its own ways.

ALLAN: What kind of things have you been doing research on?

TRENTON: Well, I just turned in my PhD dissertation on human-robot interaction.

ALLAN: Ooooh.

TRENTON: There I dealt with two separate points. One was looking at this idea of having robots in the home and then what you would do in those cases. You have to be concerned about the privacy issues that arise with having a computer with lots of sensors moving around your house.

And then the other aspect goes back to my GUI programming days at Trolltech. I was interested in animation techniques and ways that robots can move to make them a bit livelier or a bit more entertaining. I looked at how you could use animation techniques to move robots and how that affects people interacting with them.

BENEDICT: Is that related to your work with the Robot Operating System?

TRENTON: In a way. Like most things for me with FreeBSD, it's always kind of a side hustle, it seems. When I started at the University of Oslo, the Informatics department was pretty open about what you could run—whatever operating system you wanted in general. If it's a Windows machine, they, of course, will administrate everything. But if you're running a Mac or something else, you pretty much have free reign.

I said, Okay, I'm going to get one of these nice, new Thinkpads and try running FreeBSD on it. At that exact point the new Macbooks had come out and I was unimpressed with what they were offering as far as something that I could open up and repair.

BENEDICT: I hear you.

TRENTON: Having done that [opened and fixed] Macs before, I was less than confident that that wouldn't be required [again]. I got a Thinkpad and installed what was called at the time TrueOS to get the wireless card and everything working.

And I used that for about a year, and there was really no problem for the work I was doing, and then I would use virtual machines to run the Robot Operating System or ROS.

I guess I should maybe stop and explain what ROS is before going further.

ROS is a middleware that can communicate between different nodes on a network that it builds. And you can create topics that publish information, for example, pictures from a video camera, sensor readings, or things like that.

And the thing is it can be distributed so you can run it on different computers, and at the same time, it's a very standardized interface, so once you have a driver on your robot that can publish information in a good way, you can just hook it up to the node and everyone can use it. And it's an open-source project, so it's easy to get all these packages.

It's a pretty nice little system of packages and communication, and it was set up to only work on Linux. You could run it with Homebrew on the Mac and using, I guess, the Linux layer on Windows 10. But it currently was set up so ROS would do distros that followed Ubuntu.

When Ubuntu did an LTS release, ROS would do an LTS release, and they would support that on all the robots or whatever until the time ran out for Ubuntu's support period.

So I had this FreeBSD box, and I had to start working with one of these robots that was using ROS. And I thought, well, I don't want to have to do all this network hopping to get a virtu-

al machine to talk directly to this robot. And I thought I would try to port it.

I downloaded the sources and started working on building it. Almost everything is written in C++ or Python. [The build system is] all based on Cmake stuff, and as far as I could tell, most of that was already in FreeBSD [ports]. If I could follow these scripts, it would probably work okay.

I got pretty far—I got the basic communications stuff done. And I thought, okay, this might actually be something I could do. I could get this up and working because I managed to get that done in a day by just having that compile in the background while I worked on other stuff.

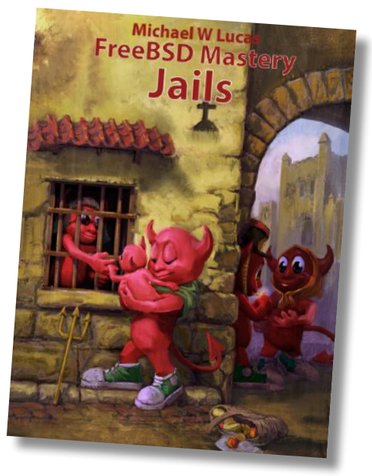
But then I realized I needed the graphical components that are also part of ROS. ROS also has a lot of debugging tools and ways of visualizing data. And simulating robots as well because you don't want to blow up a robot by putting bad code on it. You want to simulate it first.

I started to think, okay, I can start to try to port these other programs as well to get a desktop version of ROS. It was really tough because one of the first things that needed to be built was OpenCV—a Computer Vision Library. And at that time, FreeBSD [ports] had Version 2, but ROS needed Version 3.

And that was just a giant thing to be [ported and] built, and at the same time, it needed Python Qt bindings, and it didn't say for sure which ones it needed. If you know the Qt community, there are at least two different types of Qt bindings that are available for Python.

It just got to be a mess. I spent a few days on it. But then I realized I had to get a PhD done rather than spending time on getting everything to compile. Sadly, I had to bite the bullet and install Ubuntu on the computer and just go and get things done.

Then I realized, well, I could give this another shot because things have happened since I tried in 2017. We're in 2019. I saw that OpenCV [3] had come in. We had modern Qt and everything, so I figured that this could just work.



I had also read Michael W Lucas's book on jails which gave me a much better grasp on how to use jails.

I was able to create the Jail. Copy or download the sources. Install all the prerequisites and actually get compiling. And, of course, I had created ports for some of these building tools that you need for ROS, and I put those in the port's tree as well.

It was really quite nice to build ROS in jails. It got rid of issues about knowing which Qt bindings you needed and stuff like that because you had a clean environment.

ALLAN: It doesn't conflict with the version of Qt you're trying to use for your desktop at the same time?

TRENTON: You got it! I was very happy and wondering why I hadn't done this earlier. But it's difficult to just step into jails. You have to do some reading to really understand how they work.

ALLAN: Yeah, Michael Lucas discovered that when he went to write the jails book, he had to write six other books first.

TRENTON: Indeed! Indeed. And I've read those other six books. And I guess the last thing I'll say is that after reading his book I was finally able to build a CUPS Jail, which I had tried and failed multiple times previously.

ALLAN: Yeah, you might not think you need it, but once you read it, you'll realize there's all kinds of ways you could be using it to make your life easier.

TRENTON: Yeah, but back to the ROS stuff—using those jails, I was able to build up the basic communications parts again. And then I built up the desktop side of it as well. And yeah, I got a graphical ROS distro so that I could use the basic tools, the graphing tools, the visualization tools, and so on.

I still didn't have the simulators because nobody ported those yet. Those simulators are specialized for robots only. If you don't have all of ROS there, there's not much point in porting the simulators on their own.

The simulators are Gazebo, which is the big one, though as far as I know, some people are considering going over to using things like Unity or Unreal as well. But I don't think they have the same physics engines as the gazebo simulator.

Then I thought this might be an interesting talk for a EuroBSDcon. I wrote up an abstract for that and sent it off to EuroBSDcon.

BENEDICT: Was EuroBSDcon your first BSD conference?

TRENTON: Yeah, it was. The nice thing about EuroBSDcon was that it was coming to Lillehammer, Norway.

It was my first conference, and yeah, it was quite enjoyable.

ALLAN: What would you say was the thing you enjoyed the most about the conference?

TRENTON: Well, let's see. One thing I enjoyed was the tutorial the first day. That was Tom Jones's tutorial on FreeBSD hardware hacking. I had never done a lot of that stuff myself. I've always been involved with people who have, but I've never actually got to sit down and try out some of the stuff myself.

The thing for me was just to be able to hang out a bit and talk with people and learn some new things about what's going on with current developments in all of the BSDs. I think it's also kind of fun to sit and see some of those talks instead of finding them later when they're posted online. Because you also get a chance to at least ask some questions and stuff that you might not have had a chance to otherwise.

I would recommend it, especially if it's in a country that you're living in and it's not a horrible burden to get to it, I would say definitely do it. It's a nice conference. And well-run.

BENEDICT: Switching gears a little bit, since you started with the BSDs and then had a little break and then came back to it, do you have some tips or advice for people who are starting out with the BSDs, what they should do, or what they should avoid? Some pitfalls maybe?

TRENTON: Hmmm. When I started with FreeBSD, it was important to be able to use things that friends were using as well. The Internet was much different in the late '90s than it is today in terms of being able to find information.

The nice thing was that you could get real-time help right then and there. I guess I would say one thing is if you can get into BSDs with a friend, that will be helpful because then you can help one another as you discover things.

If I were to offer a tip—if you are going to use ZFS, you should really set up the zfstools port early when you get started, so you are automatically doing snapshots so that you can actually roll back, because I think it's easy to say, "Oh, you just rolled back to a snapshot." But if you forgot to take a snapshot, you're in trouble.

ALLAN: Yeah, in addition to whatever manual snapshots you might decide to make, please also make a snapshot every 15 minutes to last 4 hours and then some grandfathering test scheme.

TRENTON: Yeah. Set that up because then you can be much more fearless in what you're doing. You don't have to be too worried about breaking stuff. And then you can use boot environments as well.

ALLAN: Yeah, I guess I hadn't really thought about the fact that in the 1990s, if you were going to put FreeBSD on a machine, it was probably going to be the only OS on the machine. Or you'd fight with dual booting or whatever. But you were pretty much giving the whole machine over to it, whereas with the virtual machines you can completely risk-free spin up a FreeBSD machine without having to risk messing up your computer and having to fix it.

TRENTON: Exactly, exactly. I think the only thing that becomes a constraint then is just having enough disk space to kind of dedicate to it. If you want to install a lot of software and see how that works, you have to figure out that upfront. But otherwise, it's really nice.

When I first started to look at FreeBSD again, the first thing I did was load up Virtual Box and put a FreeBSD image on there and just tried to see how it worked so I would have an idea of what I should do before I did it for real.

ALLAN: Yeah, one option we'd like to add someday is being able to have the installer say, Oh, I see you already have a FreeBSD installed. Maybe you've broken it. Would you like us to do a fresh install as a new boot environment but leave all your old files behind so that you can still get at them? But you'll have a system that boots cleanly again.

TRENTON: Yeah! That would be nice.

BENEDICT: Yeah, more work for the future.

ALLAN: And actually, before we let you go, I had another question about the Robot OS. You just talked a little bit about it and how it's distributed. Would one robot consist of multiple computers, or is it more about making many robots work together?

TRENTON: Hmm, I guess it could be both, if you wanted. Basically, you create the nodes and each node has certain topics. A robot could have lots of nodes. In one of my research projects, I was using a turtle bot, which is a robot that has a LIDAR [light detection and ranging] sensor on top that spins around to find its location.

And it also had wheels to turn and so on, and each of those things was a topic that would say what speed the robot was going and how it was turning or moving. There was another topic that was getting the data from the LIDAR, and the whole point was that you could write another node that would say, oh, I'll listen to this information from the LIDAR and use that to figure out a map location, and then I'll use another node which has the planner for the robot and will tell the robot where to move.

You end up creating a lot of different nodes, and most of the nodes are written in either C++, Python, or Common Lisp. And in general, they'll be fast enough for what you're trying to do, as long as you're not trying to do real-time stuff. That's what ROS 2 is about.

It's an interesting system and used for a lot of robots nowadays. But like a lot of these things, it is very research-oriented, and most of the ROS stuff—at least the ROS 1.0 version—

does not consider security. It's sort of using the FTP model, where you'd go to a known port and then it gives you another UDP port for listening, but you don't know what the UDP port is.

If you have a firewall, the first thing they tell you to do is create a VPN between you and the robot so that you can talk to each other.

I believe they've tried to fix that in later versions, but I remember that was an issue here at the university because the wireless network in general needs to be closed down. Suddenly, you need to open all these ports, and you don't know what they are until you actually make a connection.

ALLAN: They kind of assign randomly when you try to connect to the robot?

TRENTON: Exactly. And that doesn't work very well. You have to create a very specific, small, closed network. But, in general, I would say most robots should probably be on a closed network [for security reasons].

ALLAN: Yeah, I guess it gets back to more of the research you were doing about how to apply privacy to the robots. If a robot has cameras for eyes and is recording what it is seeing, how do you decide what to do with that and so on?

TRENTON: Right, and there's the other question of what is actually happening with the camera images? If the camera image is something that's just being used locally, for example, if it's a depth camera, it's just trying to figure out the distance between different objects. And all that's happening is finding the robot's location and planning where it's supposed to go, which will never involve going off to the Internet. It's all being processed locally.

In that situation, it's not such a major privacy issue. There are a whole bunch of issues that need to be looked at, and that was one of them.

BENEDICT: Yeah, you'll have to leave something for the future scientists. They're still work…

TRENTON: There is plenty to do. I was at a robot conference just last week, and one of the points discussed was that there are so many research problems that have not been solved and will not be solved for a long time. No question that there will be plenty of things for the future scientists.

BENEDICT: Excellent.

ALLAN: Trenton, is there anything else you'd like to talk about before we let you go?

TRENTON: I can't really think of anything other than thanks very much for the interviews you do.

ALLAN: Thank you very much for being willing to talk with us.

2020 Events Calendar

BSD Events taking place through June 2020

BY ANNE DICKISON AND DRU LAVIGNE

SCaLE 18x • March 5–8 • Pasadena, CA

https://www.socallinuxexpo.org/scale/18x

SCaLE 18x—the 18th annual Southern California Linux Expo—will take place at the Pasadena Convention Center. SCaLE 18x expects to host 150 exhibitors, along with nearly 130 sessions, tutorials, and special events. The FreeBSD Foundation is pleased to be holding a full-day Intro to FreeBSD workshop on Friday, March 6. Also, be sure to stop by the FreeBSD booth and say hi!

FOSSASIA Summit 2020 • March 19–22 • Singapore, Singapore

https://summit.fossasia.org/

The FOSSASIA Summit, Asia's leading open-technology conference for developers, startups, and IT professionals, will take place at the Lifelong Learning Institute Singapore. Join Foundation Executive Director Deb Goodkin as she presents "FreeBSD, the Other UNIX-like Operating System and Why You Should Get Involved!"

AsiaBSDCon • March 19–22 • Tokyo, Japan

https://2020.asiabsdcon.org/

AsiaBSDCon is a conference for users and developers on BSD-based systems. The conference is for anyone developing, deploying, and using systems based on FreeBSD, NetBSD, OpenBSD, DragonFlyBSD, Darwin, and MacOS X. AsiaBSDCon, a technical conference, aims to collect the best technical papers and presentations available to ensure that the latest developments in our open-source community are shared with the widest possible audience.

BSDCan • June 3–6 • Ottawa, Canada

https://www.bsdcan.org/2020/

The 17th annual BSDCan will take place at the University of Ottawa. BSDCan hosts talks and tutorials on a range of topics based around the BSD family of operating systems. There will be a 2-day DevSummit and tutorials, followed by a 2-day conference.